A Metadata Catalog Service for Data Intensive Applications

Gurmeet Singh, Shishir Bharathi, Ann Chervenak, Ewa Deelman, Carl Kesselman, Mary Manohar, Sonal Patil, Laura Pearlman

Information Sciences Institute, University of Southern California

Marina Del Rey, CA 90292

{gurmeet, shishir, annc, deelman, carl, mmanohar, sonal, laura}@isi.edu

Abstract

Advances in computational, storage and network technologies as well as middleware such as the Globus Toolkit allow scientists to expand the sophistication and scope of data-intensive applications. These applications produce and analyze terabytes and petabytes of data that are distributed in millions of files or objects. To manage these large data sets efficiently, *metadata* or descriptive information about the data needs to be managed. There are various types of metadata, and it is likely that a range of metadata services will exist in Grid environments that are specialized for particular types of metadata cataloguing and discovery. In this paper, we present the design of a Metadata Catalog Service (MCS) that provides a mechanism for storing and accessing descriptive metadata and allows users to query for data items based on desired attributes. We describe our experience in using the MCS with several applications and present a scalability study of the service.

1 Introduction

Data intensive scientific applications promise dramatic progress in scientific discovery. These applications produce and analyze terabyte and petabyte data sets that may span millions of files or data objects. Driven by trends in storage technology, computational power and network bandwidth and the capabilities of Grid computing [1][2][3], these data intensive applications are increasing in scope and sophistication to perform scientific data collection and analysis on a scale never before achievable.

Metadata services are required to support these data intensive applications. Metadata is information that describes data. Metadata services allow scientists to record information about the creation, transformation, meaning and quality of data items and to query for data items based on these descriptive attributes. Accurate identification of desired data items is essential for correct analysis of experimental and simulation results. In the past, scientists have largely relied on ad hoc methods (descriptive file and directory names, lab notebooks, etc.) to record information about data items. However, these methods do not scale to terabyte and petabyte data sets consisting of millions of data items. Extensible, reliable, high performance Grid services are required to support registration and query of metadata information.

There are various types of metadata, and it is likely that a range of metadata services will exist in Grid environments that are specialized for different types of metadata cataloguing and discovery. Some metadata relate to the physical characteristics of data objects, such as their size, access permissions, owners and modification information. Replication metadata information describes the relationship between logical data identifiers and one or more physical instances of the data. Other metadata attributes describe the contents of data items, allowing the data to be interpreted. For example, climate modeling data sets may have associated metadata attributes that include variables such as temperature, surface pressure, precipitation and cloud cover. High-energy physics metadata might include information about the period of time during which events were detected in a particle collider. Often this descriptive metadata conforms to an ontology agreed upon by an application community. A special case of descriptive metadata is provenance information, which records how data items are created and transformed. Provenance metadata might describe what experimental apparatus, simulation or analysis software produced the data item. This provenance information can be used to track a series of analyses or transformation steps on a data item or to reproduce a data item.

Because the types of metadata that need to be recorded for data characterization and discovery are diverse and have different requirements for performance, reliability and consistency, we expect there to be multiple Grid services that are specialized for different types of metadata. For example, in our previous work [4], we presented a Replica Location Service that exclusively contains metadata information related to data replication. We took advantage of relaxed consistency requirements for this type of metadata to design a distributed replica location framework. Other types of metadata, such as storage system metadata or descriptive metadata that characterize the contents of data files, typically require stricter consistency and may be stored in centralized catalogs or in federated databases.

A key challenge for Grid designers is the correct factoring or partitioning of metadata cataloguing functionality among a collection of services and the integration or federation of those services to provide efficient data discovery.

In this paper, we present a design of a Metadata Catalog Service (MCS) that provides a mechanism for storing and accessing descriptive metadata and allows users to query for data items based on desired attributes. The main contributions of our work are the following:

- We characterize the cataloguing and discovery of descriptive or logical metadata as a distinct Grid service component, separate from services for physical storage metadata or replica metadata.
- We describe a general metadata schema.
- We provide extensions for user-defined metadata attributes, which are important for supporting application-specific metadata ontologies.
- We describe an API for storing and querying metadata.
- We describe a prototype Metadata Catalog Service implementation, which uses open source web service and relational database technology.
- We discuss our experience using the MCS in two applications, the Earth System Grid (ESG) [5] and the Laser Interferometer Gravitational-Wave Observatory (LIGO). We also discuss the use of MCS with a workflow mapping system Pegasus [6].
- We characterize the scalability of the MCS in terms of sustained queries per second as a function of various parameters such as database size, number of clients, etc.
- Finally, we discuss our current redesign of the MCS for greater flexibility, extensibility and performance.

2 The Role of Metadata Services in Grid Data Management

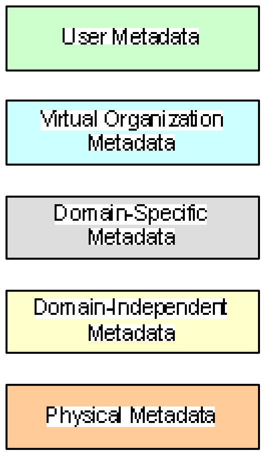

Figure 1: Metadata Types

Figure 1 shows several types of metadata that may be used to characterize data objects. At the lowest level, *physical metadata* include information about the characteristics of data on physical storage systems as well as replica location metadata. Services that maintain physical metadata include file systems and database management systems. *Logical name* attributes are identifiers for data content that may have zero or more physical instances or replicas. Replica location services provide mappings between logical name attributes and physical location information for one or more replicas.

The next level in Figure 1 shows domain-independent metadata. These are general metadata attributes that apply to data items regardless of the application domain or virtual organization in which the data sets are created and shared. Such "generic" attributes include logical names, information about the creator and modifier of data content, authorization and audit information, and information about aggregations of data objects such as collections and views.

The upper levels of Figure 1 show metadata attributes that are specific to an application domain, a virtual organization or to particular users of the data. Domain-specific metadata attributes are often defined by metadata ontologies that are developed by application communities. For example, physicists or earthquake engineers may agree on a common set of terms and metrics that are useful for characterizing shared data sets and represent these using a common set of metadata attributes. Similarly, a virtual organization that includes multiple scientific or corporate institutions may define an additional set of metadata attribute conventions for characterizing data sets. Finally, individual users may want to associate metadata attributes such as annotations with data items or collections.

In our work, we define *Metadata Services* as services that maintain mappings between logical name attributes for data items and other descriptive metadata attributes and respond to queries about those mappings. In particular, Metadata Services support domain-independent, domain-specific, virtual organization and user metadata attributes, as shown in Figure 1. We distinguish Metadata Services from those that contain physical metadata, such as Replica Location Services.

Metadata Services play a key role in the publication and the discovery and access of data sets.

Publication

Publication is the process by which data sets and their associated attributes are stored and made accessible to a user community. Many data intensive scientific applications publish data sets as a community. For example, when results of a scientific experiment are obtained, they are calibrated, put into a standard format, and made available or published to the community. Part of the publication process includes using the Metadata Service to associate domain-independent, domain-dependent and virtual organization metadata attributes with the data set. Subsequent to the publication, some members of the community may use the Metadata Service to annotate the data set with their own observations using user attributes and make these annotations available to a controlled subset of the community. Scientists may also perform analysis of published data sets to produce new data sets, which are also published along with associated metadata attributes. Other members of the community may organize published data items of interest into customized views by associating aggregation attributes with existing data sets in the Metadata Service.

Discovery and Access

Discovery is the process of identifying data items of interest to the user. The Metadata Service allows users to discover data sets based on the value of descriptive attributes, rather than requiring them to know about specific names or physical locations of data items.

Typically, the Metadata Service forms one component of data discovery and access in a Grid. Figure 2 illustrates a simple scenario for attribute-based data discovery and access using our Metadata Service, called the MCS, the Replica Location Service [4], and the GridFTP data transport protocol [7]. In this scenario, a client application first queries the Metadata Service to find data sets with particular attribute values (1). The Metadata Service responds with a list of logical name attributes for data items with matching attributes (2). Next, the client queries the Replica Location Service (3), which returns a list of physical locations for the data content identified by the logical names (4). Finally, the client selects replicas for access, contacts the storage systems where the data items reside (5), and the desired data sets are returned using the GridFTP protocol (6).

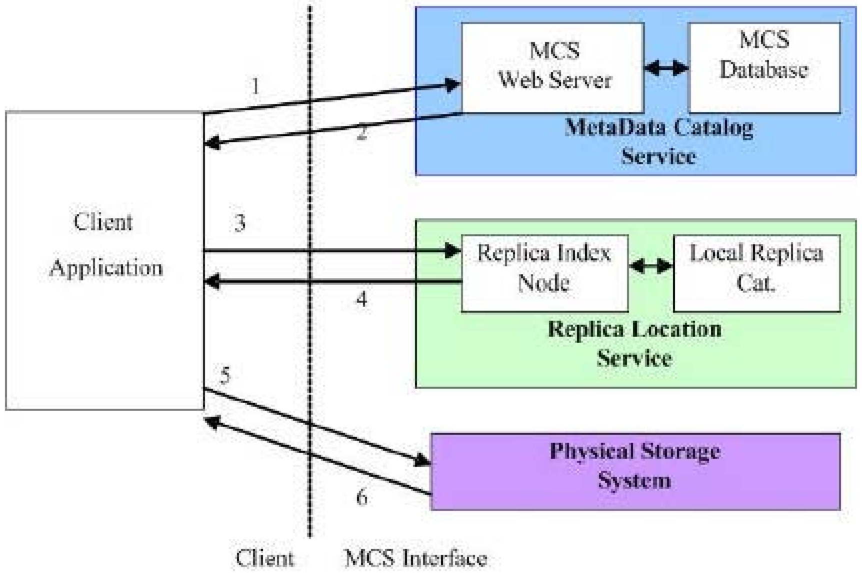

Figure 2: A usage scenario for Metadata Services in Grid environments.

3 Requirements for the Metadata Service

Next, we describe the requirements that drive the design of the Metadata Catalog Service.

First, a Metadata Service must provide a mechanism for associating logical name attributes with domain-independent metadata attributes according to a pre-defined schema as well as user-defined attributes that extend this schema to store domain-dependent, virtual organization and user metadata attributes.

Second, the Metadata Service must support queries on its contents. Based on a set of attribute values specified in a query, the Metadata Service must return a set of zero or more logical name attributes for metadata mappings that match the specified attributes and values. Metadata Service queries must also return the values of one or more additional metadata attributes associated with the logical name attribute.

The Metadata Service must implement policies regarding the consistency guarantees, authentication, authorization, and auditing capabilities provided by the service. Typically, Metadata Services maintain strict consistency over their contents, since inaccuracies in the metadata database can cause incorrect identification of data items, resulting in incorrect analyses. More relaxed consistency semantics may also be possible and are a subject of future research. Authentication and authorization allow control over who is allowed to add, modify, query and delete mappings in the Metadata Service. One of the policies implemented by a Metadata Service is the granularity of authentication and authorization operations, ranging from providing access to the entire contents of the service to restricting access on individual mappings or even attributes. Auditing information that may be maintained by the Metadata Service includes information about creators and creation times of metadata mappings as well as a log of all the accesses to a particular metadata mapping, including the identity of the user and the action that was performed.

Next, the Metadata Service may support the ability to aggregate metadata mappings into collections or views by associating aggregation attributes with logical name attributes. These aggregation attributes are useful for associating logically related metadata mappings, for example, metadata related to a set of experimental runs. The Metadata Service must allow users to associate attributes with the aggregation of mappings, to avoid requiring that every attribute be assigned to each individual mapping in the collection or view. The service must also perform access control on the aggregations of mappings.

Another requirement is that the Metadata Service should provide the ability to store attributes that describe the record of transformations on a dataset, including the data set's creation and subsequent processing. This information is sometimes called *provenance*. These transformation records may include the identity of data modifiers as well as information about analyses run and the input parameters used.

While in general, the Metadata Service avoids storing any physical metadata attributes, there may be some exceptions. To support replica management and data consistency, the Metadata Service may provide support for associating a master copy attribute with metadata mappings. A master copy is the definitive physical copy of a data item; typically, updates are made to the master copy and then propagated to other copies of the data item. The Metadata Service may also provide attributes that distinguish among multiple versions of a data item.

Metadata mappings may also contain attributes that refer to an external *container* service that is used to group together large numbers of relatively small data objects for efficient data storage and transfer. The external container service is responsible for constructing containers and extracting individual data items from the container.

Finally, the Metadata Service should provide good performance and scalability. It should provide short latencies on query and update operations and relatively high query and update rates. Performance requirements will vary by application. The Metadata Service should scale to support information about millions of data items and thousands of collections or views.

4 Components of a Metadata Service

Based on these requirements, we argue that a Metadata Service is more than simply a database service that stores metadata attributes. Rather, it is a specialized service that includes the following components:

- A data model that includes mechanisms for aggregation of metadata mappings
- A standard schema for domain-independent metadata attributes with extensibility for additional user-defined attributes
- A set of standard service behaviors
- Query mechanisms for accessing the database
- A set of standard interfaces and APIs for storing and accessing metadata
- A set of policies for consistency, access control and authorization, and auditing

5 MCS: A Metadata Catalog Service for Grids

Based on the requirements above, we present the design and implementation of the Metadata Catalog Service (MCS).

The MCS Data Model

While it could easily be extended to handle non-file data, the initial implementation of MCS assumes a file-based data model. The most basic item in MCS data model is the logical file, which is uniquely identified by a logical name. Logical collections are user-defined aggregations that can consist of zero or more logical files and/or other logical collections. Logical views are another type of aggregation that can consist of zero or more logical files, collections and/or other logical views. Figure 3 illustrates this data model.

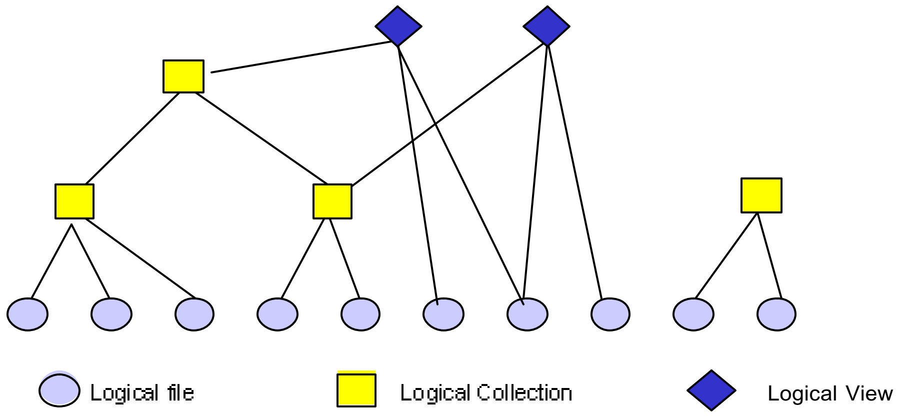

Figure 3: The Illustration of the Data Model for MCS.

One of the most important purposes of logical collections is to support authorization on groups of files rather than on individual files. If a logical file is contained in a logical collection, then the user must have appropriate permissions on both the logical file and the logical collection to access or modify the metadata associated with the logical file. To support consistent authorization, MCS requires that a logical file may belong to at most one logical collection. MCS supports a tree hierarchy of collections and simple queries that list the logical files in a collection and respond to attribute-based queries on logical collections.

To allow more flexibility for users to group files according to their interests, MCS also supports aggregations called *logical views*. A logical view may aggregate any acyclic selection of logical files, logical collections or other logical views. Logical files and logical collections may belong to many different logical views. Logical views do not affect authorization, which is enforced via access rights defined for logical collections and individual logical files. (Adding a file to a logical view is loosely analogous to creating a symbolic link in one file system directory to a file that actual resides in another directory.)

MCS Schema

Next, we briefly describe our schema design for the current implementation of the MCS. The detailed schema can be found in [9]. The attributes in the MCS schema can be divided into the following logical categories.

Logical File Metadata: Metadata attributes associated with the logical file include the following. A *logical file name* attribute specifies a name that is unique within the namespace managed by the Metadata Service. A *data type* attribute describes the data item type, for example, whether the file format is binary, html, XML, etc. A *valid* attribute indicates whether a data item is currently valid, allowing us to quickly invalidate logical files, for example, if a virtual organization determines that a logical file contains incorrect data. If data files are updated over time, a *version* attribute allows us to distinguish among versions of a logical file. If multiple versions are specified, then both the logical file name and the version number must be supplied by the client to allow MCS to uniquely identify the desired data item. A *collection identifier* attribute allows us to associate a logical file with exactly one logical collection. *Container identifier and container service* attributes allow us to specify the external container service that groups objects together and associate a container identifier with the logical file mapping. *Creator* and *last modifier* attributes record the identifications of the logical file's creator and last modifier. We also define attributes specifying *creation time* and *last modification time*. A *master copy* attribute can contain the physical location of the definitive or master copy of the file for use by higher level data consistency services. Finally, the *audit* attributes specifies whether audit information is to be recorded for this logical file.

Logical collection metadata attributes include the collection name and a description of the collection contents, which consists of the list of logical files and other logical collections that compose this collection. Each logical file can belong to at most one logical collection. Logical collections may contain other collections, but must form an acyclic collection hierarchy. In addition, the collection metadata includes a text description of the collection, information about the creator and modifiers of the collection, and audit information. Finally, there may be a parent attribute that records the identifier of the parent logical collection. MCS allows specification of an arbitrarily deep acyclic hierarchy of logical collections.

Logical view metadata attributes include the logical view name and description; information about the logical files, logical collections and other logical views that compose this logical view; attributes describing the creator and modifiers of the view, and audit information.

Authorization metadata attributes are used in addition to (or in to the absence of) an external authorization service such as the Community Authorization Service [8] to specify access privileges on logical files, collections and views. The authorization information must be maintained for individual users. If an external Community Authorization Service (CAS) is used, then authorization information must also be maintained for the CAS.

Access permissions may be defined on the MCS itself (e.g., permission to add logical files to the MCS), on a logical file (e.g., permission to modify the file's attributes), on a logical collection (e.g., permission to add a logical file to the logical collection), or on a logical view (e.g., permission to list the contents of a logical view). Access permissions specified on a logical collection apply to all logical files that comprise that logical collection (and its subcollections). The effective set of permissions on a logical file is the union of the permissions on that file and the permissions on a logical collection to which the file belongs, and so on up the hierarchy of collections.

User metadata: The MCS schema includes attributes that describe writers of metadata, including contact information. The attributes specify the distinguished name, description, institution, address, phone and email information for writers.

Audit metadata is used to record actions performed via the metadata service. An audit record includes the attributes that specify the object identifier upon which an action was performed and the object type (logical file, logical collection or logical view). Audit metadata also includes a text description of the audited action as well as the distinguished name (DN) of the user who performed the audited action. Finally, the audit record contains the timestamp at which the audited operation was performed.

User-defined metadata attributes: Extensibility of the MCS schema beyond predefined attributes is provided by allowing users to define new attributes and associate them with logical files, collections or views. Extensibility is an essential requirement, since each scientific application domain typically produces one or more metadata schemas that capture attributes of interest to that community. Users may create attributes of the following types: string, float, date, time and date/time.

Annotation attributes: MCS provides the ability for users to attach annotations to logical files, collections or views. Annotation metadata includes the identifier for the object being annotated and the object type (logical file, collection or view). The annotation attribute is a string provided by the user. Annotation metadata also includes the distinguished name of the user creating the annotation and a timestamp that records when the annotation was created.

Creation and transformation history metadata attributes record information about how a logical file was created and what subsequent transformations were performed on the data. In the current implementation, the history is a textual description of these operations and the date on which the data object was created. This information may be used to recreate the data item if it ever gets corrupted, or the application may decide to recreate the dataset if the cost of recreating it is less than the cost of retrieval. We may add more attributes to allow for more sophisticated queries, for example, queries about a particular analysis program run or experimental conditions.

External catalog metadata: Because metadata may be spread across multiple heterogeneous catalogs, we also include in the MCS schema attributes that can be used to access external catalogs. External catalog metadata attributes include the external catalog name and type (e.g., relational database) and the host name and IP address for where this external catalog may be accessed.

MCS Service Implementation

The Metadata Catalog Service uses a web service model. Its implementation uses an Apache web service front and a MySQL relational database backend. The components of MCS are shown in Figure 3. A client application program issues MCS queries using the MCS API. The MCS client sends these queries to the MCS server using SOAP. The MCS server then interacts with the MySQL database to perform the query, package the results and return them to the MCS client. Finally, the results are returned to the client application.

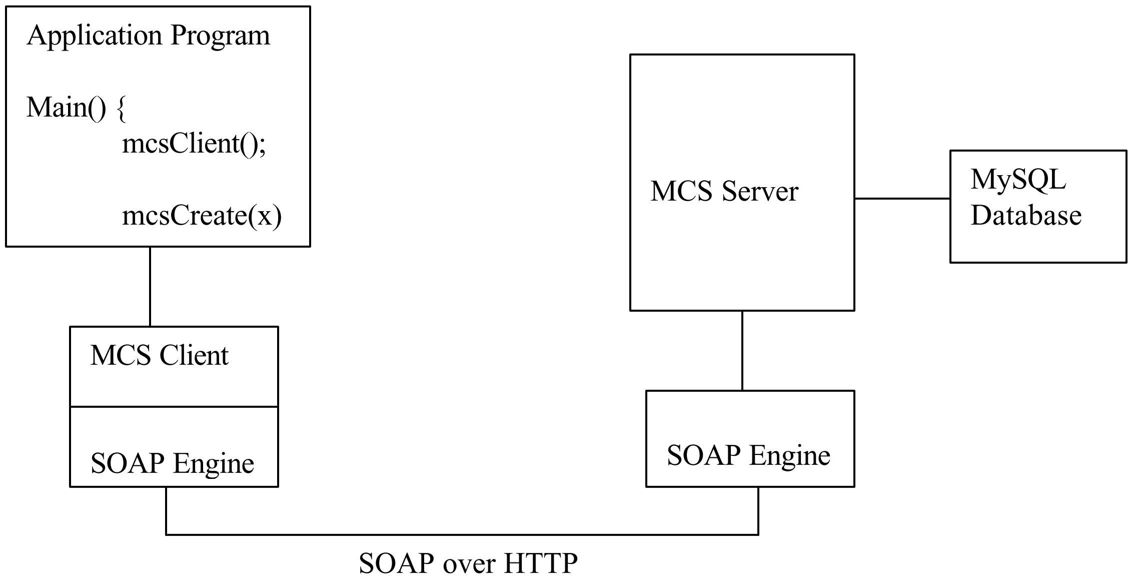

Figure 4: Overview of the Implementation.

MCS Query Mechanisms and APIs

MCS provides a synchronous java client application programming interface. For space reasons, we do not include details of the API in this paper. The client API provides the following operations:

- Querying the catalog for logical objects based on object attributes
- Querying the static attributes of a logical object
- Querying the user defined attributes of a logical object
- Querying the contents of a logical view or a logical collection
- Creating a logical file, collection or a view
- Modifying the attributes of a logical object
- Deleting a logical file, view or a collection
- Annotating a logical object
- Adding logical objects to a view

Other complex operations can be composed from these elementary operations.

Much of the client interface code was automatically generated from the WSDL description of the service using open source tools.

The MCS client issues queries using the MySQL query language to the MySQL relational database backend.

MCS Policies

The MCS provides authentication and authorization capabilities on the logical files and logical collection attributes in MCS, as described in the data model above, enabling authorized users to add, modify and delete the MCS mappings and attributes. Our design is based on the Grid Security Infrastructure (GSI). Eventually, we will integrate the MCS with the Community Authorization Service (CAS) [8, 9].

The MCS provides auditing metadata to allow recording of creation information for logical files, collections and views as well as logs of all the accesses to a particular data item, including the identity of the user and the action that was performed. MCS also provides attributes for recording information about transformations on datasets.

Finally, to support other services such as replica managers that maintain consistency among data items, the MCS provides a master copy attribute that identifies the physical location of a master copy of a file.

6 Application Experiences

We used the MCS in two applications in late 2002: the Pegasus (Planning for Execution in Grids) system [10] and its use in the Laser Gravitational Wave Observatory (LIGO) project [11] as well as in the Earth System Grid (ESG) [5] application. In this section, we summarize the lessons learned integrating MCS into the software used by these applications.

6.1 The Pegasus/LIGO Application

Pegasus [10] is a planning component developed within the GriPhyN project (www.griphyn.org) [12]. Pegasus is used to map complex application workflows onto the available Grid resources. One of the applications that uses Pegasus is LIGO. LIGO (Laser Interferometer Gravitational-Wave Observatory) is a project that seeks to directly detect the gravitational waves predicted by Einstein's theory of relativity.

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. For example, a user might request all logical files that corresponding to a particular frequency band, and the MCS will return a list of relevant files to the Pegasus planner.

Because the MCS only stores logical file names, Pegasus uses an additional component for data cataloguing and discovery, the Replication Location Service, which maps from logical file names to physical file replicas. The LIGO example shows that these two types of metadata services (MCS and RLS) can be effectively federated.

When the workflow generated by the Pegasus planner results in creation of new application data products, Pegasus uses the Metadata Catalog Service to record metadata attributes associated with those newly materialized data products. Among the data products created by LIGO analyses are time series data, frequency spectra and the results of pulsar searches. Attributes that describe these data products, including the type of data and the duration of data measurements, are stored in the MCS. To support LIGO application-specific metadata, we added 23 user-defined attributes to the pre-defined attributes provided by the MCS schema.

For the LIGO experiment, the pre-defined MCS schema plus the user-defined attributes were able to capture all the richness of the application environment. However, the next application we describe, the Earth System Grid, needed additional functionality to be integrated into the system.

6.2 The Earth System Grid Application

The MCS was loaded with climate modeling data for use in the Earth System Grid application [5]. The MCS was one component in an ESG testbed that included replica management and storage management services as well as various data storage services connected with GridFTP.

The ESG metadata followed the netCDF convention and was stored metadata in XML format. ESG scientists also stored general metadata using the Dublin Core [13] schema from the digital library community. To store ESG metadata in MCS, we added user-defined attributes to the MCS to correspond to application-specific ESG metadata attributes as well as Dublin Core attributes. Then we parsed or "shredded" the XML metadata files to extract individual attribute values and stored these.

ESG scientists successfully used the MCS to discover and query for ESG data files based on metadata attributes. Despite this success, ESG scientists observed that there was not a simple mapping between XML metadata files and MCS relational tables. Shredding of XML metadata for storage in MCS proved cumbersome and slow. Second, the ESG application did not make use of many of the general attributes that had been defined for the MCS. Third, ESG scientists wanted more flexibility in the types of queries that can be issued against the MCS.

Based on the experience of ESG scientists, we are investigating several issues for the next phase of MCS development. We are examining ways to make the MCS more easily extensible to accommodate application-specific metadata schemas more naturally. We are particularly focusing on efficient handling XML data, which is commonly used for metadata. We are studying whether a native XML database would provide better functionality than a relational database backend; however, the performance of open source XML databases is not currently sufficient to support the query rates required by ESG applications. Finally, we will provide a more general query model than the one currently provided by the MCS API.

7 Scalability of the MCS

In this section, we present a scalability study of the MCS. We experimented with databases of three sizes. For each size, we created logical collections with 1000 logical files per collection. With each logical file, 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 logical collection. We loaded databases with a total of 100,000, 1,000,000, and 5,000,000 logical files and their associated collections and attributes. Since we maintained a constant 1000 files 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.

The MCS was installed on a dual-processor 2.2 GHz Intel Xeon workstation running RedHat Linux 8.0. MCS is built upon the Apache Jakarta Tomcat 4.1.24 server and uses MySQL 4.1 relational database. We built indexes on logical file names, logical collection names and logical views (not used for these performance tests). We also built indexes on the database-assigned identifiers for these items and on (name,ID) pairs.

Our performance numbers show adds and query operations. The add operations add a logical file with ten associated user-defined attributes of various types. To maintain the size of the database, we follow each add operation with a delete operation. The simple query operation does value match for a single static attribute associated with a logical file. The complex query operation does value matches for all ten user-defined attributes associated with a logical file. The first set of performance results presented are generated using a single client host running multiple client threads. The second set of results uses multiple clients hosts, with each running multiple client threads.

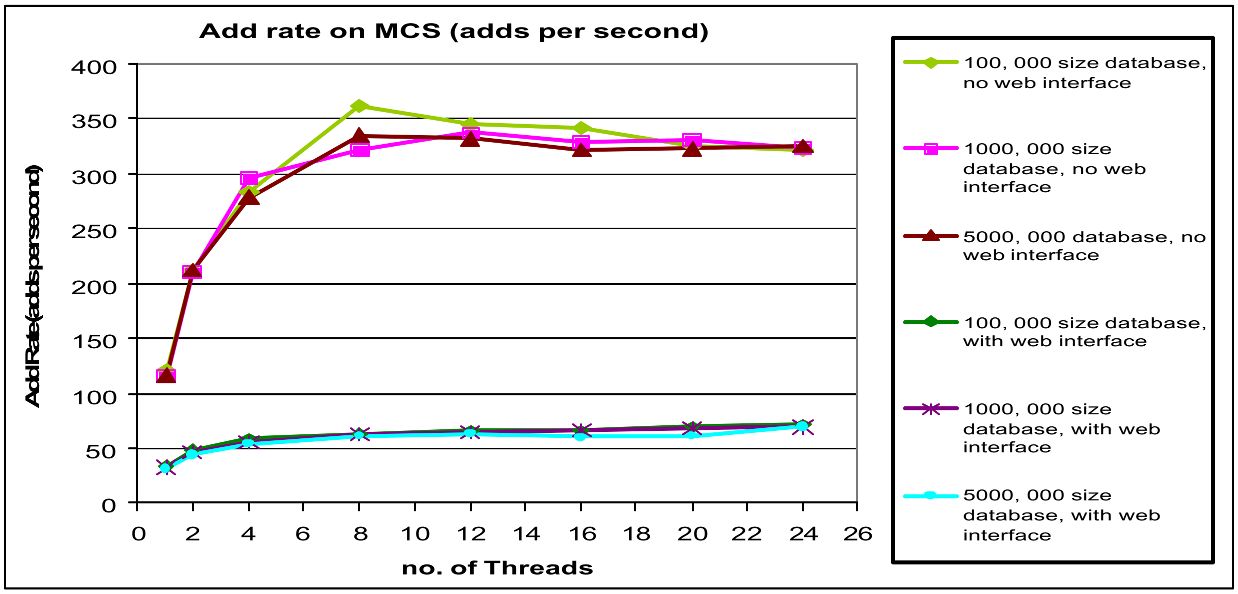

Figure 5: Add Rate on MCS with Varying Threads on a Single Client Host.

Figure 5 shows add rates per second on the MCS for the three database sizes (100,000, 1 million, and 5 million total logical files) with a single host submitting requests from a varying number of threads. As previously mentioned, these operations add logical files with ten associated attributes. The figure shows performance of both the MySQL database and of the MCS web service implementation. The MySQL performance mentioned in this section refers to the database access times and also some processing overhead for converting add and query requests into SQL statements. We use the MySQL performance numbers as the baseline for the performance of MCS. As shown, MySQL without the MCS web service interface supports add rates of approximately 360 adds per second for a database of size 100,000 logical files and approximately 340 for 1 million logical files, and 330 for the 5 million logical files.

When add requests are made through the MCS web service, the supported rates drop dramatically, to a maximum of approximately 70 per second for all database sizes. This drop in performance is due to the web service overhead incurred by MCS, making it 4.8 times slower (on the average) than the underlying database. However, just as MySQL itself, MCS scales well with increasing database size, with approximately a 7% lower add rate for the largest database size.

Figure 6 shows simple query rates from a single host with a varying number of threads for the three database sizes, directly to the MySQL database and through the MCS web service. "Simple queries" refer to MCS operations where the client queries with a single static attribute of a logical file. For the single host, the greatest performance achieved is with 12 threads. Without the web service, the MySQL database achieves a rate of over 2300 simple queries per second. When requests are made through the MCS web service, the simple query rate drops to approximately 120 queries per second, exposing again the web service overhead. In this case, the database size has little effect on the query performance.

Figure 7 shows complex query rates for a single client host submitting requests, where each query does value matches on the ten user-defined attributes associated with a logical file. For the MySQL database (without the MCS web service), the query rates are a factor of ten lower for the database of sizes of 1 and 5 million logical files compared to the database of size 100,000. With the MCS web service, the drop in complex query rates is more than 50% for the larger databases. The poor scalability of these queries is due to the complexity of matching all of the user-defined attributes.

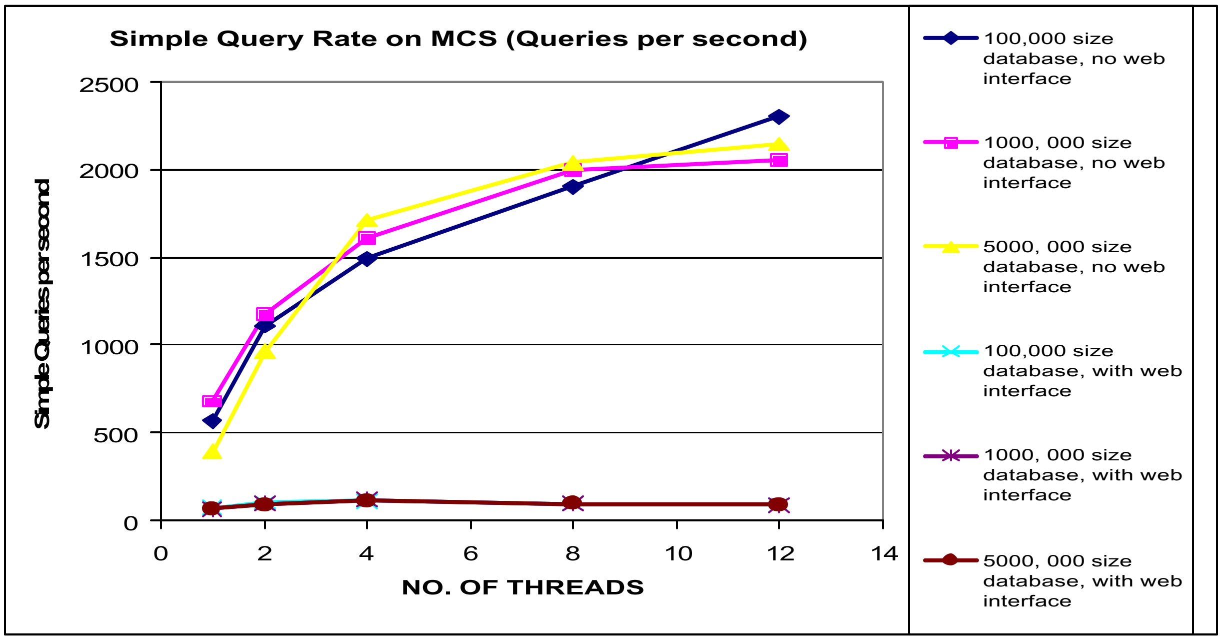

Figure 6: Simple Query Rate on MCS with Varying Threads on a Single Client Host.

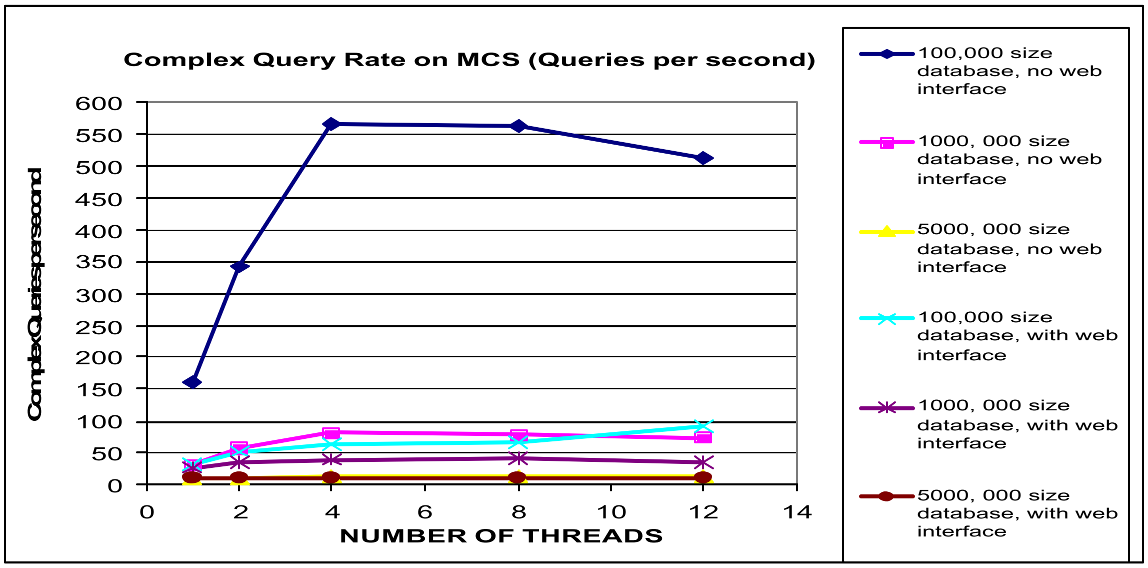

Figure 7: Complex Query Rate with a Varying Number of Threads on a Single Client Host.

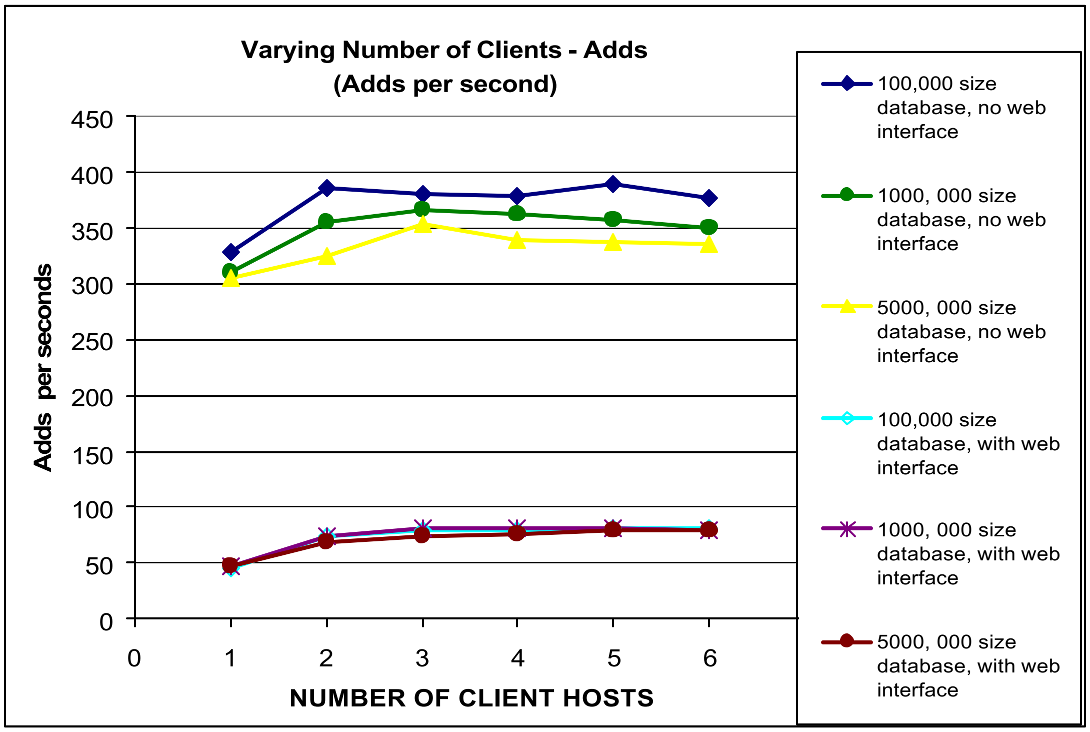

Figure 8: Add Rate with Varying Number of Hosts, Each Running 4 Threads.

The next set of performance graphs show the rates achieved when multiple clients are accessing the MCS in parallel. In Figure 8, we show add rates for requests issued by multiple hosts, with each host running four threads. For add performance, a single host submitting add requests to the MCS achieved a rate of approximately 46 adds per second for all database sizes. When up to 6 hosts are submitting add requests, the rate goes up to almost 80 adds per second for all database sizes, suggesting that the a single host cannot saturate the MCS add capabilities.

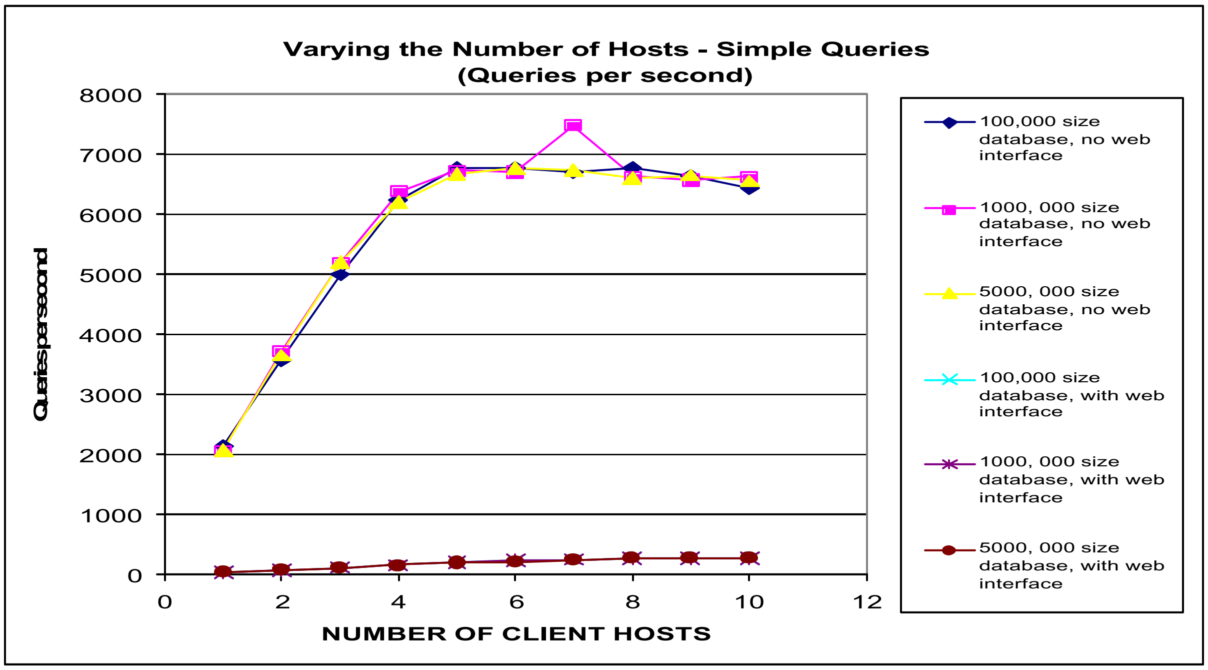

Figure 9: Simple Query Rate with a Varying Number of Client Hosts.

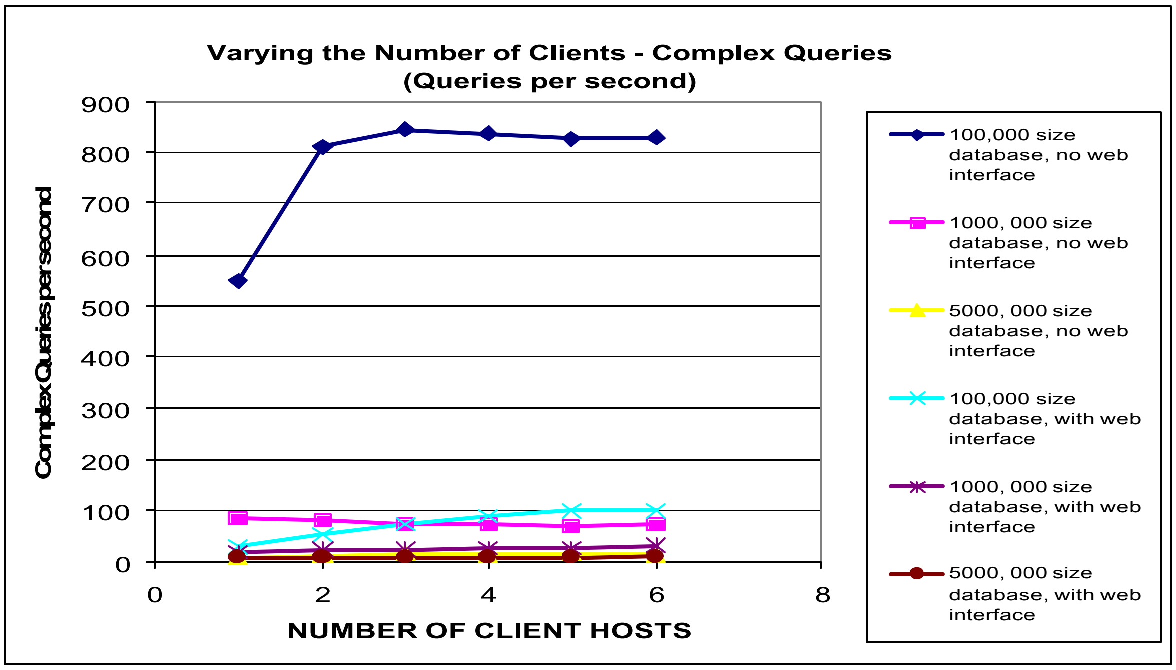

Figure 10: Complex Query Rate with a Varying Number of Hosts.

Figure 9 shows the simple query rates when we vary the number of client hosts each running 4 threads. We notice that as the number of clients increases, the performance of the MySQL database (without the web service) increases significantly, reaching a peak of almost 6,800 queries per second with 6 hosts, compared to approximately 2,000 queries per second with a single client host. The performance of MCS (with web service) sees comparatively greater improvements, increasing from 40 simple queries per second with a single host to 280 with 10 hosts. The performance of MCS may continue to improve as the number of clients increases; however, so far we have been unable to perform these measurements. Again, the size of the database does not affect the overall performance for simple queries.

Figure 10 shows the performance of MySQL and MCS when performing complex queries with a varying number of hosts. Here, we see similarities with single host performance, achieving greatest performance for the MySQL database with the smallest size (100,000 logical filenames). The performance degrades again as the database size increases, because of the overhead of searching through the large attribute space.

To better understand the effect of the number of attributes on the performance of complex queries, we have measured the performance of MySQL (without web service) as we varied the number of logical filename attributes. Figure 11 shows the results. We can see that as the number of attributes increases, the performance of MySQL degrades for all database sizes, with the most significant decrease for the 100,000 database size. The performance of MySQL drops by a factor of 3 for the 100,000 database size when the number of attributes increases from 1 to 10 and drops by approximately a factor of 4 for the 1 and 5 million database sizes.

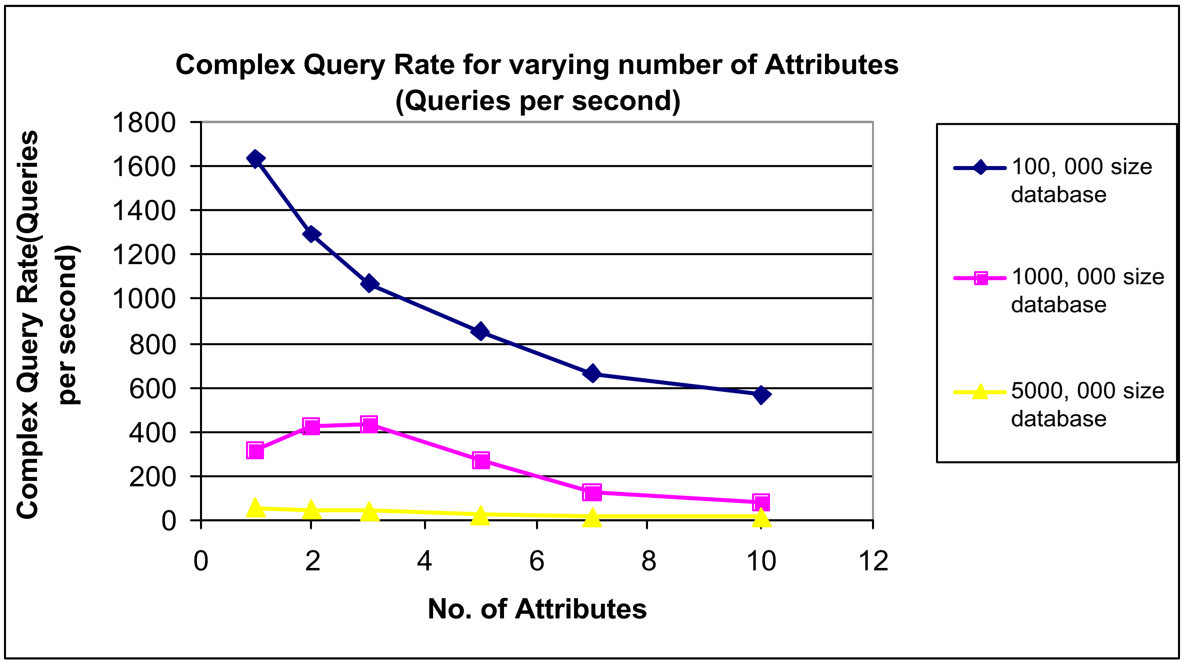

Figure 11: Complex Query Performance of MySQL as the Number of Attributes is Varied.

To summarize our performance numbers, we demonstrated that using the web service to interface to the MySQL database introduces significant performance overhead. However, even with the web service overhead, MCS scales well for many of the supported operations as we increase the size of the database. The only case where we saw relatively poor scalability is the case of complex queries, where the match is performed on the values of all 10 attributes of the logical filename. However, MCS's performance degradation is largely due to the penalty suffered by MySQL.

8 Related Work

The schema of MCS owes a great deal to the MCAT Metadata Catalog [13] of the Storage Resource Broker (SRB) from the San Diego Supercomputing Center [14]. Some of the attributes and ideas found in the MCS metadata schema have direct parallels in MCAT. For example, both MCAT and MCS support a logical name space that is independent of physical name space; both provide GSI authentication; both allow specification of a logical collection hierarchy; and both support the notion of containers to aggregate small files.

However, the Metadata Catalog Service differs from MCAT in significant 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 the MCS to be one component in a layered, composable Grid architecture. We have factored this Grid architecture so that the Metadata Catalog Service contains only logical metadata attributes. By design, the MCS is an orthogonal component, separate from other types of metadata services, such as replica location services.

The RepMec (Replica Metadata) catalog developed by the European DataGrid's Reptor project [15] is similar in its design and function to MCS. The RepMec catalog is built upon the Spitfire database service. The RepMec catalog stores logical and physical metadata. Among other functions, this catalog is used within the EDG project to map from user-provided logical names for data items to unique identifiers called GUIDs. RepMec is used in the Reptor system in cooperation with a replica location service.

9 Summary and Future Directions

We have presented the design and implementation of a Metadata Catalog Service and described our early experiences using the MCS with two applications. Based on the lessons learned from this implementation, we have undertaken a reevaluation and redesign of the Metadata Catalog Service. We plan to make the service more extensible and to provide a more general query model. As part of this redesign, we are investigating the use of other database backend technologies, including the use of a native XML database. We plan to make the next implementation of MCS an OGSA-compliant Grid service, and to that end, we have begun an implementation of the MCS using the OGSA Database Access and Integration (DAI) service. DAI provides a Grid service front end with the choice of a relational or native XML database backend.

Authorization and authentication need further attention in MCS. Although we have modeled integration of the MCS with the Community Authorization Service [8], this integration still needs to be implemented.

Finally, we plan to address the performance and reliability problems associated with a centralized MCS implementation. Until now, we have assumed that strict consistency is required for a distributed MCS implementation and have assumed that we would eventually replicate the MCS over a small number of sites to improve performance and reliability. We plan to examine the possibility of using a looser consistency model to federate self-consistent metadata catalogs, using a scheme similar to that employed in the Replica Location Service [4] and the Monitoring and Discovery Service [16]. In this model, consistent local catalogs use soft state update mechanisms to send periodic summaries of metadata discovery information to aggregating index nodes. Clients query these indexes to discover desirable data sets across a collection of metadata services and then issue subqueries to the underlying local catalogs.

Acknowledgments

This research was supported in part by the National Science Foundation 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] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann, 1999.

[2] I. Foster, "Grid Computing," presented at Advanced Computing and Analysis Techniques in Physics Research (ACAT), 2000.

[3] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of High Performance Computing Applications*, vol. 15, pp. 200-222, 2001.

[4] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B, Schwartzkopf, H, Stockinger, K. Stockinger, B. Tierney, "Giggle: A Framework for Constructing Sclable Replica Location Services," presented at SC2002, Baltimore, MD, 2002.

[5] ESG, "The Earth Systems Grid." http://www.earthsystemgrid.org

[6] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, A. Arbree, R. Cavanaugh, K. Blackburn, A. Lazzarini, and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments," *Journal of Grid Computing*, vol. 1, pp. 25-39, 2003.

[7] B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Leigh, A. Sim, A. Shoshani, B. Drach, D. Williams, "High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies," presented at SC2001, 2001.

[8] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, "A Community Authorization Service for Group Collaboration.," presented at IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.

[9] A. Chervenak, E. Deelman, C. Kesselman, L. Pearlman, and G. Singh, "A Metadata Catalog Service for Data Intensive Applications," GriPhyN technical report, 2002-11 2002.

[10] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman, "Pegasus: Planning for Execution in Grids," GriPhyN 2002-20, 2002.

[11] A. Abramovici, W. E. Althouse, and e. al., "LIGO: The Laser Interferometer Gravitational-Wave Observatory (in Large Scale Measurements)," *Science*, vol. 256, pp. 325-333, 1992.

[12] E. Deelman, K. Blackburn, P. Ehrens, C. Kesselman, S. Koranda, A. Lazzarini, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, and R. Williams., "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists," presented at 11th Intl Symposium on High Performance Distributed Computing, 2002.

[13] MCAT, "MCAT - A Meta Information Catalog (Version 1.1)."

[14] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC Storage Resource Broker," presented at Proc. CASCON'98 Conference, 1998.

[15] Guy, L., P. Kunszt, E. Laure, H. Stockinger, K. Stockinger (2002). Replica Management in Data Grids. Global Grid Forum 5.

[16] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, "Grid Information Services for Distributed Resource Sharing," presented at Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), 2001.