

Database Support for Data-driven Scientific Applications in the Grid *

Sivaramakrishnan Narayanan, Tahsin Kurc, Umit Catalyurek, Joel Saltz

Dept. of Biomedical Informatics
The Ohio State University
Columbus, OH, 43210
{krishnan,kurc,umit,jsaltz}@bmi.ohio-state.edu

Abstract

In this paper we describe a services oriented software system to provide basic database support for efficient execution of applications that make use of scientific datasets in the Grid. This system supports two core operations: *efficient selection of the data of interest from distributed databases* and *efficient transfer of data from storage nodes to compute nodes for processing*. We present its overall architecture and main components and describe preliminary experimental results.

1 Introduction

Earlier efforts in developing the Grid infrastructure focused on support for applications that required high computing power. In recent years, we have observed an explosion of data driven applications that require support for distributed storage and processing. Examples include applications that synthesize and process results generated by large scale simulations, applications that generate data products from high energy physics experiments, and applications that analyze large datasets from medical imagery. The need for distributed storage and processing in these applications arise for several reasons. First, in many cases datasets are generated at multiple sites. For instance, investigation of alternative production strategies for optimizing oil reservoir management requires large numbers of simulations using detailed geologic descriptions. Such ensembles of simulations can be run at multiple sites and results are stored on local storage systems at those sites. Second, analysis of data involves integration of multiple types of data. Output from reservoir simulations, for example, is combined with seismic datasets to better predict the reservoir properties and drive new simulations. Third, processing of data employs computationally expensive operations, which demand execution on parallel machines.

The Grid offers scientists an ideal platform to execute their compute- and data-intensive applications. The Grid infrastructure consists of heterogeneous collections of computation and storage *nodes* interconnected through local-area and wide-area networks. PC clusters built from low-cost, commodity items are increasingly becoming widely used in many areas of science, medicine, and engineering. With fast CPUs and high-speed interconnects, they provide cost-effective *compute nodes* in the Grid for computation intensive applications. With high-capacity, commodity disks, they create *active storage nodes* that enhance

*This research was supported by the National Science Foundation under Grants #ACI-9619020 (UC Subcontract #10152408), #EIA-0121177, #EIA-0203846, #ACI-0130437, #ACI-9982087, Lawrence Livermore National Laboratory under Grant #B500288 and #B517095 (UC Subcontract #10184497).

a scientist's ability to store large-scale scientific data. We refer to such storage systems as *active storage systems* because not only do they provide a platform for data storage and retrieval, but also some application-specific processing can be carried out on the system. In general, a compute node is configured to have large memory space, high speed interconnect, and high computing power to support computationally demanding applications, whereas an active storage node is designed to provide large disk-based storage space, at the expense of computing power and memory space, to support databases of large scale scientific data.

While it is relatively inexpensive and easy to build hardware platforms to provide the needed computing and storage power, developing programming and runtime support for data-driven applications in a Grid environment is a challenging problem. Support is needed to efficiently extract the data of interest, move it from storage nodes to compute nodes, and process it on a parallel or distributed machine. Processing of data involves operations and data structures that are specific to a particular application. However, we argue that runtime support can be provided for data subsetting and data movement.

In this work we develop a services oriented software middleware system, referred to as *GridDB-Lite*, to provide basic database support for efficient execution of data-driven applications that make use of scientific datasets in the Grid. This infrastructure supports two core operations.

1. *Selection of the data of interest from datasets distributed among disparate storage systems.* The data of interest is selected based on either the values of attributes or ranges of attribute values. The selection operation can also involve joins between multiple datasets and user-defined filtering operations.
2. *Transfer of data from storage systems to compute nodes for processing.* After the data of interest has been selected, it can be transferred from storage systems to processor memories for processing by a data analysis program. If the analysis program runs on a cluster, the system supports partitioning of data elements in an application-specific way among the destination processors.

In this paper we describe the main components of this middleware and present preliminary experimental results.

2 Application Model

In this section, we describe several application scenarios that motivate the design of the middleware. We then represent the common data access pattern in data-driven applications as a database query.

2.1 Application Scenarios

Oil Reservoir Management. Numerical simulation of oil and gas reservoirs can aid the design and implementation of optimal production strategies. A major challenge is incorporating geologic uncertainty. In a typical study, a scientist runs an ensemble of simulations (also called realizations) to study the effects of varying oil reservoir properties (e.g., permeability, oil/water ratio, etc.) over a long period of time. The Grid enables the execution of such simulations on distributed collections of high-performance machines. This approach can lead to large volumes of output data generated and stored at multiple locations. Analysis of this data is key to achieve a better understanding and characterization of oil reservoirs. Common analysis scenarios involve queries for economic evaluation as well as technical evaluation, such as determination of representative realizations and identification of areas of bypassed oil. Examples of the queries include “*Find the largest bypassed oil regions between time T_1 and T_2 in realization A.*” and “*Retrieve the oil saturation values at all mesh points from realizations A and B between time steps T_1 and T_2 where the production rates of wells in realization A are equal to those of the corresponding wells in realization B; visualize the results*”.

Cancer Studies using MRI. Imaging studies are increasingly viewed as having great potential in the diagnosis and staging of cancer and in monitoring cancer therapy. In Dynamic Contrast Enhanced MRI studies [64], for example, the underlying premise is the discrimination of abnormal tissue from healthy tissue by differences in microvasculature and permeability. State-of-the-art research studies involve use of large datasets, which consist of time dependent, multidimensional collections of data from multiple imaging sessions. In a distributed environment, analysis studies could be aggregated from multiple participating institutions and involve multiple image datasets containing thousands of 2D and 3D images at different spatial and temporal resolutions. In a typical study, a researcher retrieves data from longitudinal studies from multiple different imaging sessions and carries out visualization and statistical analysis (e.g., texture analysis) on the data. An example client request can be described as “*Retrieve images obtained between dates $Date_1$ and $Date_2$ from datasets $Study_1$, $Study_2$, and $Study_3$; execute a texture analysis algorithm on each dataset.*”

Water Contamination Studies. A simulation system for examining the transport and reaction of chemicals in bays and estuaries consists of a hydrodynamics simulator and a chemical transport simulator. For each simulated time step, each simulator generates a grid of data points to represent the current status of the simulated region. For a complete simulation system, the chemical transport simulator needs to be coupled to the hydrodynamics simulator, since the former uses the output of the latter to simulate the transport of chemicals within the domain. As the chemical reactions have little effect on the circulation patterns, the fluid velocity data can be generated once and used for many contamination studies. The grids used by the chemical simulator may be different from the grids the hydrodynamic simulator employs. Therefore, running a step of chemical transport simulation requires retrieving the hydrodynamics output that fall within the time step of chemical simulation from the appropriate hydrodynamics datasets stored in one or more databases, and projecting it onto the grid used by the chemical simulator. In this case, a typical query into the databases of hydrodynamics output could be “*Retrieve the velocity values between time steps T_1 and T_2 and interpolate the values to the grid used by the chemical simulator.*”

In many scientific application, as in these example applications, analysis of data is performed by 1) selecting the data of interest from the dataset, 2) transferring the selected data items from storage units to processing nodes, and 3) performing application-specific processing on the data elements. Efficient support is needed for data subsetting and data transfer to speed up the overall execution of data analysis. In the next section we describe a formulation of data subsetting patterns in scientific applications as a database query.

2.2 Data Subsetting Model

The example applications described in Section 2.1 come from very different domains. As a result, datasets and data processing structure in these applications show different characteristics. This difference is most apparent in the format of datasets and data analysis functions applied to these datasets. For instance, medical images are stored as TIFF, JPEG or DICOM files, whereas output from oil reservoir simulations is stored as a set of files, each of which corresponds to the values of a variable (e.g., oil saturation, water pressure) at each grid point over all the time steps. MRI studies involve image processing algorithms, while oil reservoir studies carry out operations to find bypassed oil pockets and operations to compare output from two simulations. In order to provide common support, we need a level of abstraction that will separate application specific characteristics from the middleware framework. The data subsetting model that is employed in this work is based on three abstractions: *virtual tables*, *select queries*, and *distributed arrays (or distributed data descriptors)*. The first two abstractions are based on object-relational database models [80]. Developers of object-relational databases represent their datasets as tables, as in relational databases. However, unlike relational databases, table rows or columns may contain user-defined complex attributes and queries can include user-defined functions, in addition to standard relational SQL statements.

```

SELECT < Data Elements >
FROM Dataset1, Dataset2, ..., Datasetn
WHERE < Expression > AND < Filter(< Data Element >) >
GROUP-BY-PROCESSOR ComputeAttribute(< Data Element >)

```

Figure 1: Formulation of data retrieval steps as an object-relational database query.

```

SELECT simA.mesh_values, simB.mesh_values
FROM simA, simB
WHERE (simA.time_step >= T1 AND simA.time_step <= T2)
      AND (simB.time_step >= T1 AND simB.time_step >= T1)
      AND (simB.prod_rate = simA.prod_rate)
      AND (simB.well_id = simA.well_id)
GROUP-BY-PROCESSOR HilbertCurvePartitioning(mesh_point)

```

Figure 2: Example query for analysis of data in oil reservoir management studies.

Virtual Tables. Datasets generated in scientific applications are usually stored as a set of flat files. However, a dataset can be viewed as a table. The rows of the table correspond to data elements. A data element consists of a set of attributes and attribute values. Data attributes are specific to application domain and can be spatial coordinates, simulated or measured values such as saturation, temperature, color, or velocity. We define two types of attributes. A simple attribute corresponds to a single scalar value. A composite (or complex) attribute consists of a list of simple attributes. For example, the corners of a mesh cell in 3D space are composite attributes as each corner is a 3D point. However, variables such as gas pressure or oil saturation are simple attributes.

Select Queries. With an object-relational view of scientific datasets, the data access structure of an application can be thought of as a *SELECT* operation as shown in Figure 1. Data elements selected by the *SELECT* operation are grouped based on a computed attribute. In the figure, the *< Expression >* statement can contain operations on ranges of values and joins between two or more datasets. *Filter* allows implementation of user-defined operations that are difficult to express with simple comparison operations.

Distributed Arrays. The client program that carries out the data processing can be a parallel program implemented using a distributed-memory programming paradigm such as MPI [77], KeLP [36], or High-performance Fortran (HPF) [48]. In these paradigms, data elements are distributed among processors to achieve parallelism. The partitioning of data elements can be represented as a *distributed array*, where each array entry stores a data element. This abstraction has been successfully used in HPF and the CHAOS libraries [24, 25, 70, 69] for data partitioning and in MetaChaos [30, 29, 67] for exchange of distributed data structures between two parallel programs. This abstraction is incorporated into our model by the *GROUP-BY-PROCESSOR* operation in the query formulation. *ComputeAttribute* is another user-defined function that generates the attribute value on which the selected data elements are grouped together based on the application specific partitioning of data elements.

As an example, the second query in the oil reservoir management scenario can be expressed as shown in Figure 2. In the figure, *simA* and *simB* represent the tables that store simulation output from realizations A and B. The query selects the scalar values over all the mesh points at time steps between T_1 and T_2 from the two realizations. Values are returned from the time steps at which the production rates of wells in realization A are equal to the production rates of wells in realization B. In this example, a Hilbert curve

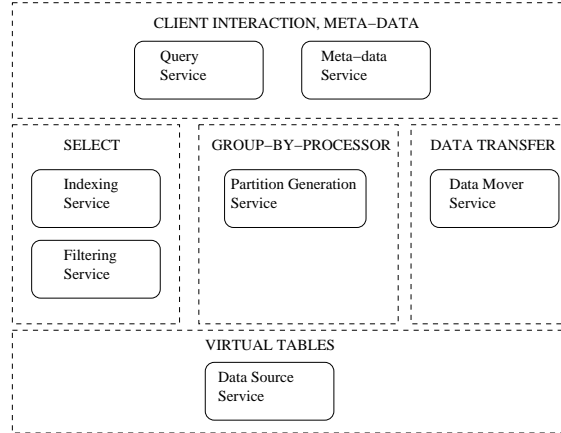


Figure 3: GridDB-Lite Architecture.

based partitioning function [34, 57] is used to compute the group-by-processor attributes.

3 System Support

In this section we describe a middleware framework, called GridDB-Lite, that is designed to provide support for 1) **select operations**: subsetting operations that can involve range queries and join operations over scientific datasets distributed across multiple storage nodes, 2) **group-by-processor operations**: partitioning of the selected data elements onto a data structure distributed across compute nodes, and 3) **data transfer operations**: copying of the selected data elements to their destination processors. GridDB-Lite is organized as a set of coupled services as shown in Figure 3.

This architecture is based on the design of two frameworks we have developed; Active Data Repository and DataCutter. The *Active Data Repository* [19, 35] (ADR) is an object-oriented framework designed to efficiently integrate application-specific processing of data with the storage and retrieval of multi-dimensional datasets on a parallel machine with a disk farm. ADR consists of a set of modular services, implemented as a C++ class library, and a runtime system. Several of the services allow customization for user-defined mapping and aggregation operations on data. An application developer has to provide accumulator data structures, which are used to hold the intermediate results, and functions that operate on *in-core* data to implement application-specific processing of *out-of-core* data. The runtime infrastructure provides support for common operations such as index creation and lookup for range queries, management of system memory, and scheduling of data retrieval and processing operations across a parallel machine. *DataCutter* is a component-based middleware framework designed to provide support for subsetting and user-defined processing of large multi-dimensional datasets across a wide-area network [7, 8, 53]. It provides an indexing service and a filtering service. The indexing service supports accessing subsets of datasets via multi-dimensional range queries. The filtering service supports the filter-stream programming framework for executing application-specific processing as a set of components, called *filters*, in a distributed environment. Processing, network and data copying overheads are minimized by the ability to place filters on different platforms. In designing the software infrastructure, we build on the services-oriented architecture of ADR and the distributed execution model of DataCutter.

3.1 Client Interaction, Meta-Data

The **query service** provides an entry point for clients to submit queries to the database middleware. It is responsible for parsing the client query to determine which services should be instantiated to answer the query. The query service also coordinates the execution of services and the flow of data and control information among the services.

The **meta-data service** maintains information about datasets, and indexes and user-defined filters associated with the datasets. A scientific dataset is composed of a set of data files. The data files may be distributed across multiple storage units [58, 33, 54]. The meta-data for a dataset can, for example, consist of a user-defined type for the dataset (e.g., MRI data, satellite data, oil reservoir simulation data), the name of the dataset, the list of attributes and attribute types (e.g., integer, float, composite) for a data element in the dataset, and a list of tuples of the form $(fi\ lename, hostname)$, where *fi lename* is the name of the data file and *hostname* is the name of the machine on which the data file is stored.

3.2 Virtual Tables

The representation of a dataset as a table is suitable for formulating queries, but the actual storage of the dataset may be much different from this representation. Datasets in scientific applications are usually stored in flat files, and file formats vary widely across different application domains. Either a dataset should be converted into a common representation as tables and stored in the database in that format, or *virtual tables* should be created when the dataset and its elements are accessed. With the second approach, applications can access their data directly when desired. The **data source** service provides a view of a dataset in the form of virtual tables to other services. It provides support for implementing application specific *extraction* objects. An extraction object returns an ordered list of attribute values for a data element in the dataset, thus effectively creating a virtual table.

3.3 Support for Select

Support for efficient execution of select operations is provided by two services. The **indexing service** encapsulates indexes for a dataset. For example, an index can be built to serve range queries [26, 19, 35]: Given a range query, it returns a list of tuples that contain information for data elements that intersect the query bounding box. A select query may contain user-defined filters and attributes that are not indexed. To support such select operations, GridDB-Lite provides a **filtering service** that is responsible for execution of user-defined filters. A filter takes a subset of the attributes of a data element and returns a scalar value (e.g., integer, float, boolean). The filtering service is responsible for execution of the $\langle Expression \rangle$ statements in Figure 1. A query can involve join operations between two or more datasets. The filtering service is also responsible for execution of join operations.

3.4 Support for Group-by-processor

After the set of data elements that satisfy the query has been determined, the data should be partitioned among the processing units of the compute nodes. The distribution of data among processors can be described by a distributed data descriptor as defined, for example, by KeLP [36] and HPF [48]. Alternatively, a common data partitioning algorithm can be employed. The purpose of the **partition generation service** is to make it possible for an application developer to export the data distribution scheme employed in the data analysis program. If the partition is described by a distributed data descriptor, which can be attached to the query submitted to the system, the partition generation service should be able to interpret the data descriptor. In some cases, the distribution of data among the processors can be expressed in an application-specific compact form. In those cases, the partition generation service computes parameters for a partitioning function that is invoked during data transfer to client.

3.5 Data Transfer

The selected data elements should be extracted from the dataset and copied to the destination processors, on which the client data analysis program executes, based on the partitioning computed by the partition generation service. The **data mover** service provides a *planner* function that computes an I/O and communication schedule. This schedule orders the retrieval of data elements from storage units to achieve high I/O bandwidth and minimize communication overheads. The Data Mover service provides *data movers*, which are components that execute in a distributed environment. The data movers are responsible for actually moving the data elements to destination processors.

3.6 Execution of a Query

In GridDB-Lite, query execution is carried out in two phases; an *inspector* phase and an *executor* phase. The *inspector/executor* model [69] has been successfully applied to provide runtime and compiler support for irregular concurrent problems. In this work, the goal of the inspector phase is to perform the steps needed to compute a schedule for moving the data from storage nodes to destination processors. The idea is that an analysis program may want a particular type of partitioning of data among the destination processors, but the amount of data obtained and distribution of data values will not be known a priori. An inspection phase is carried out to determine those values so that the client program can allocate space on client processors to receive and store the selected data elements. The executor phase carries out the plan generated in the inspector phase to move data elements from source machines to the destination processors.

Step 1. Upon receiving a query, an index lookup is performed. Index returns information needed to extract tuples from each data source by extraction objects in the data source service. We classify tuples into three categories: 1) attributes that are used to determine whether the select predicate is satisfied (*select attributes*), 2) attributes that are used to determine how the query result will be partitioned among the destination processors (*partition attributes* or *group-by-processor attributes*), and 3) other attributes that are part of the result returned to the client program (*result attributes*).

Step 2. The extraction object generates a table that associates each extracted tuple with a unique ID, the values of select attributes, and the values of partition attributes. Extraction objects are executed in parallel on data sources where the dataset is stored. The unique ID incorporates the identity of each data source. The table generated by extraction objects is referred to as *unfiltered planning table*.

Step 3. Unfiltered planning table entries are streamed to the filtering service to determine which tuples satisfy the select predicate. The select filters executed by the filtering service remove the tuples that do not satisfy the predicate; they also strip the values of the select attributes. After the filtering operation, a table, referred to as *filtered planning table*, is created that contains only tuple unique IDs and partition attributes.

Step 4. Filtered planning table is streamed from the filtering service to the partition generation service to determine the partitioning of result tuples among the processors of the client program. The partition generation service associates each unique ID with a processor or computes parameters for a partitioning function to be invoked by the data mover service. The partitioning function determines the destination processor of each result tuple. We should note that the partition generation service can employ a data-independent partitioning strategy, such as block cyclic distribution of data elements. In that case, it is not needed to send the filtered planning table to the partition generation service.

Step 5. Information from the partition generation service and the filtering service is sent to the data mover service. The data mover service uses this information to compute an I/O and communication schedule and move data. The data mover service makes use of the extraction objects of the data source service to extract the result attributes of the selected tuples. The unique IDs associated with selected tuples are used by extraction objects to scan the data sources for the selected elements. Note that data may go to processor memories or to storage units. In the former case, the client program should provide pointers to memory

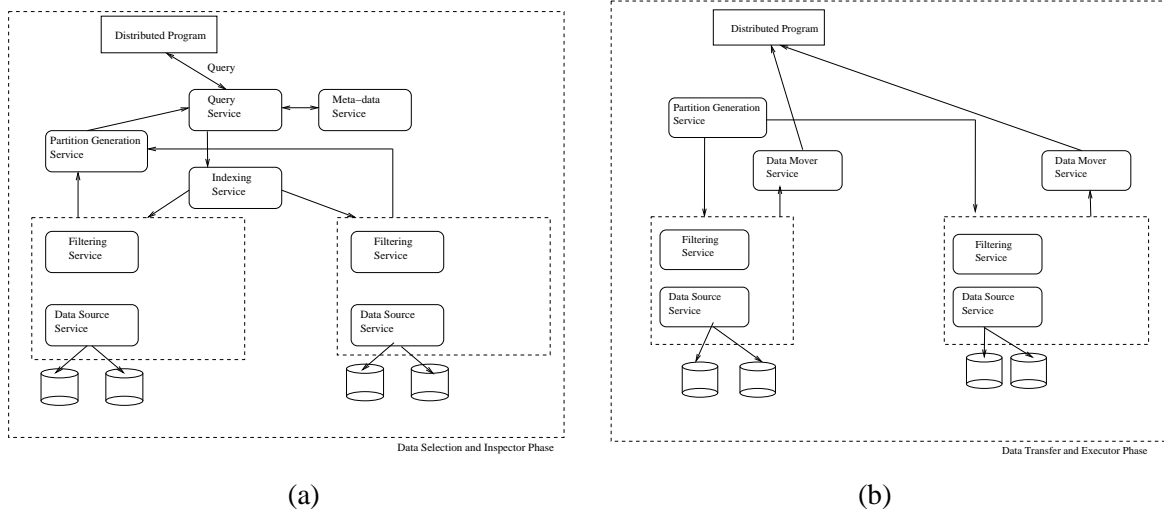


Figure 4: (a) The inspector phase. (b) The executor phase.

buffers on each of the destination processors. In the latter case, the client program should provide a list of file names on each destination processor.

Steps 1, 2, 3, and 4 correspond to the inspector phase, while step 5 is performed in the executor phase. Figure 4 shows a possible instantiation of the services and the execution of a query through the inspector and executor phases.

3.7 Speeding up Query Evaluation

Several optimizations can be employed to speed up the execution of queries. The filtering service can be co-located with the data source service on the machines where the dataset is stored in order to reduce the volume of wide-area data transfer. If the partition generation service does not require data aggregation to determine partitioning of tuples, it can be co-located with the filtering and data source services on the same processor, cluster, or SMP. If the partition generation service computes parameters for a partitioning function, then data should be streamed from the filtering service to the partition generation service, but that service should return only the parameters for the partitioning function.

The inspection phase described in Section 3.6 is carried out using individual data elements. A performance drawback of this approach is that the number of data elements can be very large, resulting in long execution times. In many scientific applications, datasets can be partitioned into a set of data chunks, each of which contains a subset of data elements. Each chunk also can be associated with metadata. Examples of metadata would be a spatial bounding box. If a dataset is partitioned into data chunks, the inspection phase can be carried out using the data chunks. In this approach the execution steps for evaluating a query are as follows.

Step 1. Upon receiving a query, an index lookup is performed. Index returns meta-data associated with data chunks along with information needed to extract tuples from each data source by extraction objects in the data source service.

Step 2. Meta-data associated with data chunks are streamed to the partition generation service to determine the partitioning of data among the processors of the client program. The partition generation service associates each chunk meta-data with processor(s)¹ or computes parameters for a partitioning function to be

¹The set of data elements contained in a data chunk may be partitioned among multiple processors. Therefore, a data chunk may be associated with more than one processors.

invoked by the data mover service.

Step 3. Information from the partition generation service and the filtering service is sent to the data mover service. The data mover service uses this information to compute an I/O and communication schedule and move data.

Step 4. The data mover service makes use of the extraction objects of the data source service to extract the result attributes of the selected tuples. The extraction object generates a table that associates each extracted tuple with the values of select attributes. Extraction objects are executed in parallel on data sources where the dataset is stored.

Step 5. Table entries from extraction objects are streamed to the filtering service to determine which tuples satisfy the select predicate. The select filters executed by the filtering service remove the tuples that do not satisfy the predicate; they also strip the values of the select attributes. After the filtering operation, a table, referred to as *filtered result table*, is created.

Step 6. The filtered result table is streamed from the filtering service to the data mover service. The data mover service transfers the selected data elements to the client nodes according to the partitioning information.

These steps are similar to those executed when the inspector phase is done using data elements. However, the inspector phase is executed in steps 1 and 2. The executor phase spans steps 3 – 6. We should note that when using data chunks for inspection, no data elements are extracted during the inspection phase. Thus, *upper bounds* on how many data elements will be sent to each processor and how much space will be needed at each processor are determined at the end of the inspection phase. The I/O and communication schedule is computed using this information by the data mover service. Some metadata information can be used for each chunk to compute tighter upper bounds on the number of data elements and distribution of data elements. For instance, a second level of spatial index can be maintained to describe spatial distribution of data elements in each chunk and use this to estimate how many data elements in a chunk would be needed to satisfy a spatial range query. Metadata could also contain a list of number of data elements with various attribute value ranges.

The services presented in this paper are designed to address core functionality required to support the efficient execution of data subsetting and data movement operations. The infrastructure does not provide such services as security, resource allocation, resource monitoring. We plan to develop interfaces to middleware infrastructures that have been developed to provide those services. These infrastructures include Globus [41], which provides support for resource discovery and resource allocation across multiple domains, the Storage Resource Broker [79] which provides unix-like I/O interface to distributed collections of files across a wide-area network, and the Network Weather Service [87] which can be used to gather on-the-fly information about resource performance and availability in a distributed environment.

4 Prototype Implementation

In order to investigate the implementation and performance issues, we have developed a prototype of the services infrastructure described in this paper. The prototype is implemented on top of DataCutter [7, 8, 11]. The current implementation supports indexing operations for range queries, user-defined filtering of data, and data transfer from storage clusters to compute clusters across local- and wide-area networks.

4.1 Indexing Service

The prototype indexing service is implemented using the indexing service of DataCutter. The DataCutter indexing service employs R-tree based indexing methods [4] to provide a multi-level hierarchical indexing scheme implemented via *summary index files* and *detailed index files*. The elements of a summary index file

associate metadata (i.e. an MBR) with one or more data chunks of the dataset and/or detailed index files. Detailed index file entries themselves specify one or more data chunks. Each detailed index file is associated with some set of data files, and stores the index and other metadata for all data chunks in those data files.

4.2 Filtering Service

We employed the filtering service of the DataCutter infrastructure in the prototype implementation. The DataCutter filtering service implements a filter-stream programming model for developing and running filters. Filters are connected via *logical streams*. A *stream* denotes a uni-directional data flow from one filter (i.e., the producer) to another (i.e., the consumer). A filter is required to read data from its input streams and write data to its output streams only. The overall processing can be realized by a *filter group*, which is a set of filters connected through logical streams. Filter groups allows a network of filters to be applied on data, thus making it possible to compose complex filtering operations from single filters. When a filter group is instantiated to process data, the runtime system handles the TCP/IP connections between filters placed on different hosts. Filters placed on the same host execute as separate threads.

4.3 Data Source Service

The data source service employs read filters that use unix I/O functions for disk-based storage systems. These filters read data from storage units and pass them to other filters through streams. Application-specific extraction objects can be implemented as DataCutter filters. An extraction filter reads a data buffer from its input stream, extracts the data elements, and transfers them to the next filter through streams.

4.4 Partition Generation Service

This service is implemented using the DataCutter filtering service. An application developer can implement the application specific data partitioning as a filter. This filter receives data element meta-data from the indexing and filtering services of DataCutter via streams and partitions the data among processors of the destination machine. The default implementation of this provides support for hilbert-curve based partitioning of data [58].

4.5 Data Mover Service

In the prototype implementation, a filter, referred to as *planner filter*, is used to implement the planner function of the data mover service. A copy of the planner filter is instantiated on each processor that stores a portion of the dataset. The planner filter sorts the data elements, stored locally on that processor, with respect to their file offsets to minimize I/O overheads. Data movers also are implemented as DataCutter filters. Data mover filters are connected to read filters implemented in the data source service. They can be instantiated on multiple processors on storage clusters.

4.6 Query Service and Query Execution

Applications implemented using DataCutter consists of filters and a console process. The console process is used to interact with clients and coordinate the execution of application filters. In our prototype, when a query is received, first the indexing service is invoked. After index lookup, the user-defined filters specified in the query and the read filters of the data source service are instantiated on the machines where the datasets are stored, and indexing results are streamed to those filters. The data partitioning filters are connected to the output streams of the indexing service and user-defined filters. These filters form a filter group that is

executed in the inspector phase. After the inspector phase is completed, the read filters of the data source service, the planner filter and data mover filters of the data mover service are instantiated. These filters form the filter group to move the data elements selected by the query to the destination machines.

5 Related Work

Requirements associated with the need to access geographically dispersed data repositories have been a key motivation behind several large data grid projects. Biomedical Informatics Research Network (BIRN) [12] and Shared Pathology Informatics Network (SPIN) [78] are examples of projects that target shared access to medical data in a wide-area environment. BIRN initiative focuses on support for collaborative access to and analysis of datasets generated by neuroimaging studies. It aims to integrate and enable use of heterogeneous and grid-based resources for application-specific data storage and dissemination. GriPhyN project [44] targets storage, cataloging and retrieval of large, measured datasets from large scale physical experiments. The goal is to deliver data products generated from these datasets to physicists across a wide-area network. The objective of Earth Systems Grid (ESG) project [31] is to provide Grid technologies for storage, publishing, and movement of large scale data from climate modeling applications. The TeraGrid [81] effort aims to develop an integrated systems software suite capable of managing a scalable set of grid based data and compute resources. The EUROGRID project [32] intends to develop tools for easy and seamless access to High Performance Computing (HPC) resources. The BioGrid component of the project will provide a uniform interface that will allow biologists and chemists to submit work to HPC facilities without having to worry about the details of running their work on different architectures. MEDIGRID [56] is another project recently initiated in Europe to investigate application of Grid technologies for manipulating large medical image databases. The Asia Pacific BioGRID [3] is an initiative that aims to provide an Asia Pacific wide computational resource package that will benefit all biomedical scientists in the Asia Pacific. BioGrid plans to extend the scope of an earlier project, APBioNet, to include a comprehensive range of software services.

In order to harness wide-area network of resources into a distributed computation and data management system, these large data grid projects should be layered on robust and efficient middleware systems. As a result, a large body of research has been focused on developing middleware frameworks, protocols, and programming and runtime environments. These efforts have led to the development of middleware tools and infrastructures such as Globus [41], Condor-G [39], Storage Resource Broker [79], Network Weather Service [87], DataCutter [8, 10, 9], Legion [43], Cactus [2], and Common Component Architecture [17], and many others [62, 1, 22, 86, 71, 16, 82, 83]. Our software infrastructure can leverage these middleware tools.

The Logistical Networking tools developed at the University of Tennessee provide support for network storage technologies for large scale data storage and communication [55]. These tools include internet backbone protocol, distributed storage infrastructure, eXnode, and L-Bone. Our middleware services can make use of support provided by these tools for management of and access to remote storage. For example, the data source service can be layered on the logistical tools to access datasets remotely. Moreover, optimizations such as data caching can take advantage of network storage by using logistical networking tools. The middleware framework described in this paper does not directly address issues pertaining to physical storage of the datasets, but provides services designed for data subsetting, filtering, and partitioning operations on large datasets. In this respect, GridDB-Lite is complementary to the logistical networking tools.

As grid computing has become more ubiquitous and with the emergence of Web services technologies, we are witnessing a shift towards a services oriented view of the Grid. An Open Grid Services Architecture (OGSA) [38, 37] has recently been proposed. OGSA builds on Grid and Web services [18, 42] technologies required to create composite information systems and to address standard interfaces and definitions for creating, naming, and integrating Grid services.

A number of runtime support libraries and compiler support have been developed to carry out the pre-processing and data movement needed to efficiently implement compute-intensive scientific algorithms on distributed-memory machines and networks of workstations. There has been a lot of work on optimizing the execution of *irregular* applications through a combination of compiler analysis and run-time support [25, 45, 46, 47, 59, 88]. In these applications, input and output are usually different fields of data items of the same dataset, which is often represented as a graph. A reduction loop is used to update values stored at the vertices of the graph with values from their neighboring vertices, sweeping through edges of the graph one edge per iteration. Most of the previous work focused on *in-core* applications, in which data structures fit in aggregate main memory space on homogeneous, distributed-memory machines. The main goal is to partition the iterations among processors to achieve good load balance with low induced interprocessor communication overhead. This is achieved by first partitioning the dataset into subsets with few edges connecting the subsets, and then assigning iterations to processors accordingly. An iteration can be assigned to a processor that owns the data elements to be updated in the iteration (i.e. the *owner computes* rule [47, 49]), or to the processor that owns the maximum number of data elements referenced in the iteration [63].

A few projects developed support for *out-of-core* applications on systems where disks are locally attached to processing nodes. Brezany et. al [13] extend the *inspector-executor* approach, pioneered by the CHAOS run-time system [25] for distributed-memory machines, for out-of-core irregular applications. More recently, acknowledging no single strategy always outperforms the others, Yu and Rauchwerger [88] developed a decision-tree based run-time system for shared-memory machines that employs a library of parallel reduction algorithms, selecting the one that best supports the data reference pattern in a program.

Several run-time support libraries and parallel file systems have been developed to support efficient I/O in a parallel environment [6, 21, 51, 52, 61, 73, 84, 20, 15, 65, 68, 74, 75]. These systems mainly focus on supporting regular strided access to uniformly distributed datasets, such as images, maps, and dense multi-dimensional arrays. Our work, however, focuses on efficiently supporting subsetting and data movement operations over subsets of large datasets in a Grid environment.

Parallel database systems have been a major topic in the database community for a long time [27, 60, 14]. Much attention has been devoted to the parallel implementation of relational joins [60, 72] and spatial joins [14, 50]. The database management community has also developed federated database technology to provide unified access to diverse and distributed data. However, most of the efforts have been angled toward relational databases. There are a few middleware systems designed for remote access to scientific data. Distributed Oceanographic Data System (DODS) [28] is one such example. DODS allows access to scientific datasets from remote locations. DODS is similar to our framework in that it allows access to user-defined subsets of data. However, our framework differs from DODS in that GridDB-Lite builds on a more loosely coupled set of services that are designed to allow distributed execution of various phases of query evaluation (e.g., subsetting and user-defined filtering).

There are some recent efforts to develop Grid and Web services implementations of database technologies [23, 40]. Raman et. al. [66] discuss a number of *virtualization* services to make data management and access transparent to Grid applications. These services provide support for access to distributed datasets, dynamic discovery of data sources, and collaboration. Bell et. al. [5] develop uniform web services interfaces, data and security models for relational databases. The goal is to address interoperability between database systems at multiple organizations. Smith et. al. [76] address the distributed execution of queries in a Grid environment. They describe an object-oriented database prototype running on Globus and MPICH-G. Our GridDB-Lite framework is targeted towards applications that make use of large, mostly read-only, scientific datasets and is designed to efficiently provide the core functionality needed for querying and data transfer.

6 Experimental Results

In this section we present preliminary performance results for the prototype implementation. We look at the performance figures for the inspector and executor phases. The hardware configuration used for the experiments consists of two Linux PC clusters. The first cluster, referred to here as **OSUMED**, is made up of 24 Linux PCs and hosted at the Ohio Supercomputer Center. Each node has a PIII 900MHz CPU, 512MB main memory, and three 100GB IDE disks. The nodes are inter-connected via Switched Fast Ethernet. We used 16 nodes of this cluster for the experiments. The second cluster, referred to here as **DC**, is located at Biomedical Informatics Department at Ohio State University. The DC cluster is composed of 5 Pentium 4 1.8GHz processors. Each node has 512MB memory and 160GB disk space and is connected to other nodes by Switched Fast Ethernet. The two clusters are connected to each other over a 100MBit wide-area network.

In the first set of experiments, we used an application emulator based on a satellite data processing application. Processing the data acquired by satellite-based sensors is a key for Earth Sciences [19]. An AVHRR satellite dataset consists of a number of measurements by satellite orbiting the earth continuously. Each measurement is a data element and is associated with a location (latitude, longitude) on the surface and the time of recording. Five sensor values are stored with each data element. Therefore, a data element in an AVHRR dataset can be viewed as having 8 attributes (two spatial, one time dimension, and five sensors). A typical analysis processes satellite data for up to a year and generates one or more composite images of the area under study. A query specifies a rectangular region and a time period. Generating a composite image requires projection of the globe onto a two dimensional grid; each pixel in the composite image is computed by selecting the “best” sensor value that maps to the associated grid point. An application emulator preserves the important data, computation, and communication characteristics of the application, but allows the input and output sizes to be scaled [85]. For the experiments, we generated a dataset corresponding to 60 days of AVHRR satellite data. The total size of the dataset is 60GB. The dataset was divided into about 55000 data chunks, each of which contains 36000 data elements. The dataset was distributed across a subset of the nodes on OSUMED and DC using hilbert curve based declustering. The queries in the experiments covered the entire surface of the earth, but requested data for a time period of one hour. This query resulted in about 2.5M elements (a total of 80MB).

Figure 5 shows the execution time of different phases in query execution. These phases are *Index Lookup* (Step 1 in Section 3.6), *Filtering and Partition Generation* (Steps 2-4), and *Data Transfer* (Step 5). In these experiments, we varied the number of nodes on which the dataset is distributed. Data caching was not employed. The dataset was stored on OSUMED, and the client program ran on 8 nodes of the same machine. As is seen from the figures, index lookup time remains almost fixed, as the indexing service was instantiated only on one node in these experiments. Our results show that filtering and partition generation and data transfer steps scale well when the dataset is distributed across more nodes. In the experiments, we instantiated data extraction and filtering components on all the storage nodes. The partition generation service employed a fixed partitioning of the result image into vertical strips irrespective of the input data. Hence, communication of the filtered planning table to the partition generation service was not necessary in this case.

In order to examine the performance of the prototype implementation in a distributed environment, we declustered the dataset across DC and OSUMED. In the experiments, we varied the storage system configuration by using X nodes from OSUMED and Y nodes from DC for storing the dataset. The client program was executed on 8 nodes of OSUMED. Figure 6 shows the execution time of various phases, when the number of storage nodes is varied. As is seen from the figure, we observe a similar trend to that of Figure 5. We should note that the timing values in this experiment are slightly better than the timing values when only the OSUMED nodes are used for storing the data. The main reason behind this improvement is the fact that the DC nodes are much faster than the OSUMED nodes. Hence, the extraction filters execute

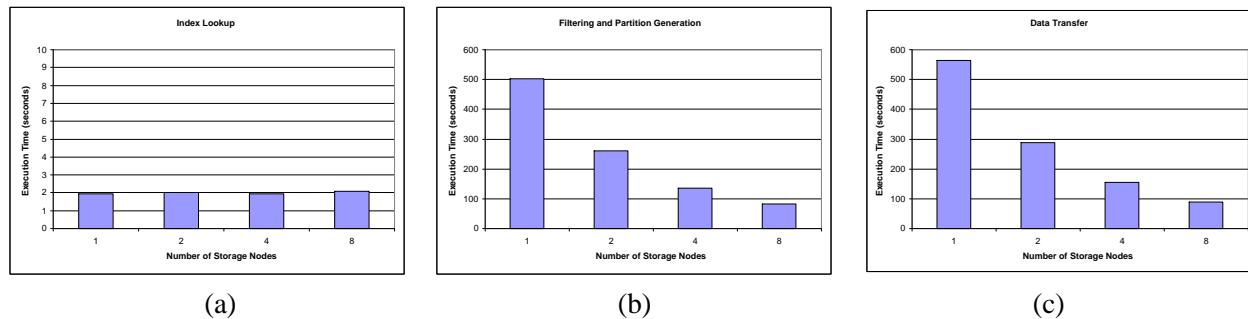


Figure 5: Execution time of various phases of query evaluation with varying number of storage nodes. (a) Index lookup. (b) Filtering and Partition Generation. (c) Data Transfer to client. The client was run on OSUMED where the dataset was stored.

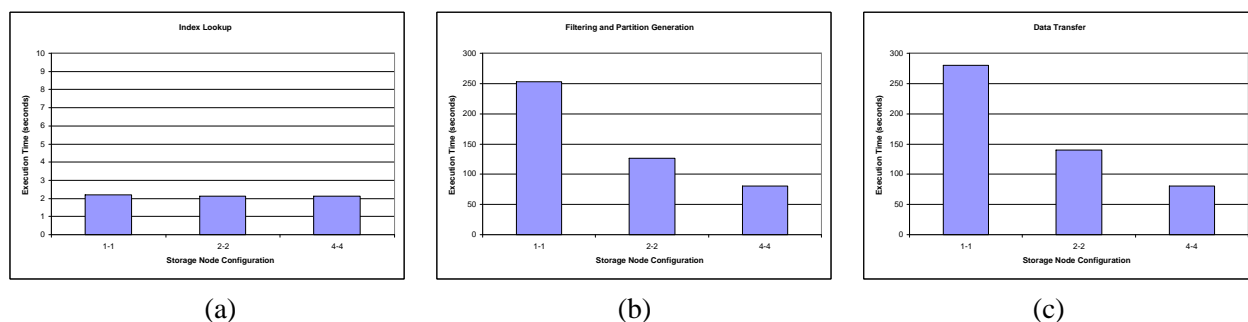


Figure 6: Execution time of various phases of query evaluation with varying storage node configurations. In the graphs, X-Y denotes the number of nodes used for storage on OSUMED and DC, respectively. (a) Index lookup. (b) Filtering and Partition Generation. (c) Data Transfer to client. The client was run on 8 nodes of OSUMED.

faster on the DC nodes, thus reducing the total execution time. Figure 7 shows the execution time of extraction filters on one node of DC and OSUMED.

Figure 8 shows the execution time of the various phases of query evaluation when the size of the input dataset is varied. We scaled up the input dataset from 60GB to 420GB. The 420GB dataset corresponds to about 420 days of AVHRR satellite data. The datasets were distributed across 8 nodes on OSUMED using hilbert curve based declustering. We used the same query for each dataset. The query covered the entire surface of the earth, but requested data for a time period of one hour. The result of the query was about 2.5M elements (a total of 80MB). As is seen from the figure, the execution time of the index lookup phase increases as the dataset size is increased. This is expected as the number of data chunks in the dataset, hence the number of entries in the index, increases with more satellite data. As a result, index search takes longer time. The execution times of the other two phases, on the other hand, remain almost the same across the three different datasets. This is because, since we used the same query for all the datasets, the number of chunks (and data elements) that intersect the query is the same. Hence, after the index lookup operation, the filtering, partition generation, and data transfer phases process the same number of data elements for each dataset.

In the second set of experiments we investigate the impact on performance of using data chunks in the inspection phase (see Section 3.7). We used a 16Kx16K 2-dimensional array of 20-byte data elements as the input dataset. The elements of this array were grouped into square chunks. A query submitted to the system is divided into 16x16 subregions. Each subregion is assigned to a processor of the client program.

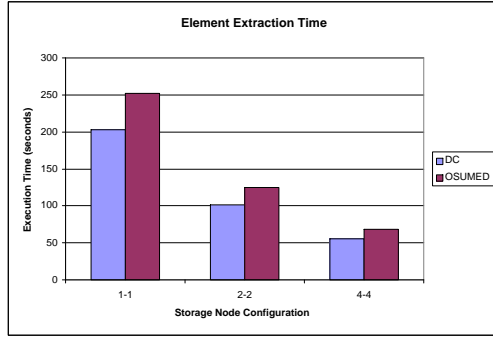


Figure 7: Execution time of the extraction filter on one node of DC and OSUMED.

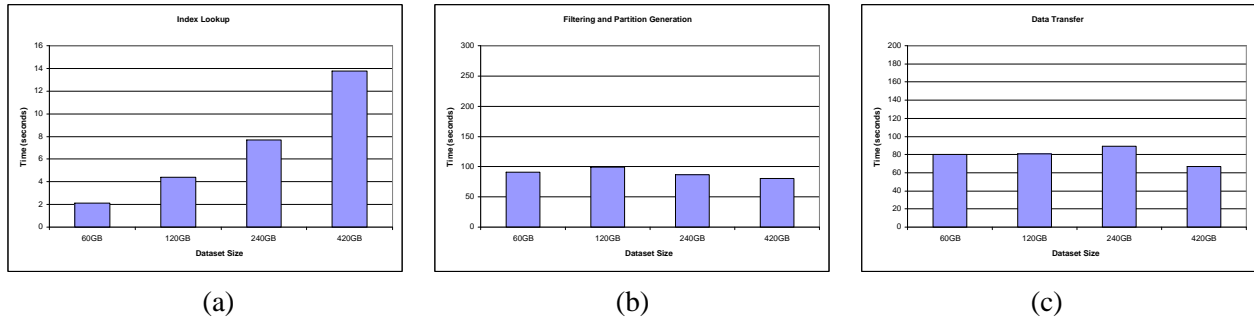


Figure 8: Execution time of the various phases of query evaluation with varying the size of the dataset. The dataset is stored on 8 nodes. (a) Index lookup. (b) Filtering and Partition Generation. (c) Data Transfer to client. The client was run on OSUMED where the dataset was stored.

In the experiments, the subregions were randomly assigned to client processors using uniform distribution. If a data element falls into subregion i , the element is sent to the corresponding client processor. Since a data chunk contains a group of data elements, a data chunk may intersect multiple subregions. Figure 9 shows the execution times of the inspector phase when the inspector phase is carried out using data chunks and using data elements, respectively. In these experiments, we varied the size of a chunk from 128×128 to 512×512 elements; the query requested 1024×1024 region of the array. The experiments were performed on OSUMED. The input dataset was declustered across 4 processors in the system and the client program, to which the data elements were transferred, ran on 8 processors – the client program was implemented as a set of filters which received data elements and discarded them as efficient execution of the client program is out of the scope of this paper.

As is seen from Figure 9, the inspector phase using data chunks takes very little time (0.2 – 0.5 seconds) compared to the case when data elements are used in that phase (about 120 seconds). This is mainly because of the fact that a data chunk encompasses multiple data element. As a result, the number of data chunks that intersect a query is much smaller than that of data elements. The results show that if the inspector phase is to be executed multiple times before the executor phase is carried out, the inspector phase should be done using data chunks to minimize the overall execution time. For example, there are multiple data partitioning strategies that can be used for some application domains (e.g., recursive bisection, graph partitioning, and hypergraph partitioning methods for irregular applications). A data analysis application may want to run several data partitioning algorithms to, for instance, obtain the most load balanced distribution of data for a given query. In that case, the inspector phase is performed multiple times. Note that, as described in Section 3.7, if the inspector phase is executed using chunks, only upper bounds on how many data elements

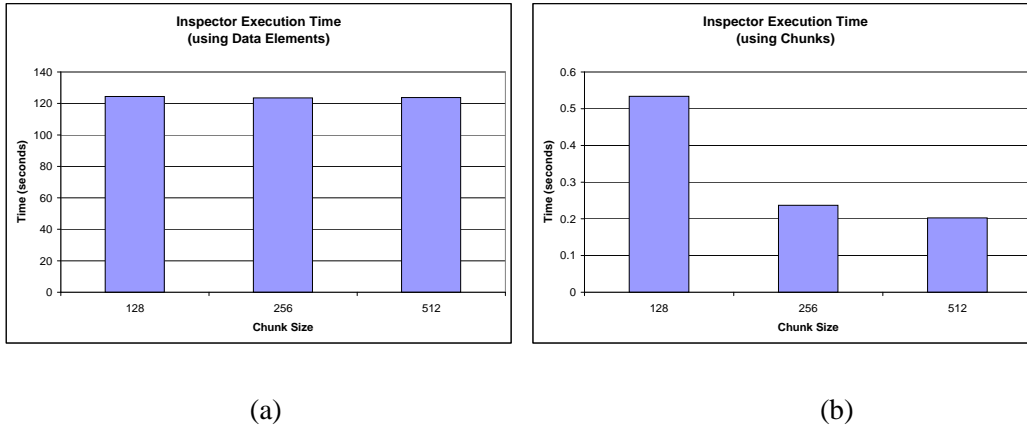


Figure 9: The execution time of the inspector phase. (a) When the inspector phase is carried out using data elements. (b) When the inspector phase is carried out using data chunks.

will be sent to each processor and how much space will be needed at each processor are determined at the end of the inspection phase.

7 Conclusions

In this paper we describe a middleware that is designed to provide support for 1) *Efficient selection of the data of interest from datasets distributed among disparate storage systems*, and 2) *efficient transfer of data from storage systems to compute nodes for processing* in a Grid environment. We also present preliminary experimental results using a prototype implementation of the infrastructure.

The middleware is proposed and designed to address issues that pertain to a key step in large scale data analysis, which is the *extraction of the data of interest from large, distributed datasets*. As dataset sizes continue to grow, an efficient solution to this step will be increasingly important to wide-scale deployment of data analysis in the Grid. Our approach is to develop a set of coupled services that collectively support the steps needed to associatively search distributed datasets on storage clusters and efficiently transfer selected data elements to compute clusters for processing. Building the middleware support as a set of coupled services instead of a tightly integrated system has several advantages. First, this makes it easier to develop customized instances of the middleware for different applications. An application developer can make use of instances of services developed by other application developers and for other applications. Second, this organization is more suitable for execution in a distributed, heterogeneous environment. Each service can be executed on a different platform to minimize computation and communication overheads.

While this paper has focused on a subset of applications from engineering and medical research that describe datasets in time dependent two- or three-dimensional space, there is a rapidly growing set of other applications in science, engineering, and medicine that make use similar datasets. The techniques and underlying framework should be applicable to those applications. Moreover, the proposed middleware framework and techniques build on the notion of virtual tables and the abstraction of data access in scientific applications as a database query. Therefore, we anticipate that applications that can formulate their access patterns as SQL queries and present their datasets as one or more virtual tables should be able to benefit from our middleware and techniques.

We plan to extend this work in several directions. First, we will implement several real applications using the prototype implementation to further examine performance tradeoffs and evaluate design decisions. Second, we will develop and evaluate methods for queries that involve join operations among multiple

datasets. Third, we plan to develop an object-relational frontend for query formulation. For this purpose, we plan to adapt the front end provided by PostgreSQL. Fourth, we will implement a Web services version of the infrastructure to have standard service interfaces and client APIs.

References

- [1] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *Proceedings of IEEE Mass Storage Conference*, April 2001.
- [2] G. Allen, T. Damlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting efficient execution in heterogeneous distributed computing environments with Cactus and Globus. In *Proceedings of the 2001 ACM/IEEE SC01 Conference*. ACM Press, Nov. 2001.
- [3] Asia Pacific BioGrid. <http://www.apgrid.org>.
- [4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R^* -tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM-SIGMOD Conference*, pages 322–331, May 1990.
- [5] W. H. Bell, D. Bosio, W. Hoschek, P. Kunszt, G. McCance, and M. Silander. Project spitfite - towards grid web service databases. <http://www.cs.man.ac.uk/grid-db/documents.html>.
- [6] R. Bennett, K. Bryant, A. Sussman, R. Das, and J. Saltz. Jovian: A framework for optimizing parallel I/O. In *Proceedings of the 1994 Scalable Parallel Libraries Conference*, pages 10–20. IEEE Computer Society Press, Oct. 1994.
- [7] M. Beynon, T. Kurc, A. Sussman, and J. Saltz. Design of a framework for data-intensive wide-area applications. In *Proceedings of the 9th Heterogeneous Computing Workshop (HCW2000)*, pages 116–130. IEEE Computer Society Press, May 2000.
- [8] M. D. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz. DataCutter: Middleware for filtering very large scientific datasets on archival storage systems. In *Proceedings of the Eighth Goddard Conference on Mass Storage Systems and Technologies/17th IEEE Symposium on Mass Storage Systems*, pages 119–133. National Aeronautics and Space Administration, Mar. 2000. NASA/CP 2000-209888.
- [9] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed processing of very large datasets with DataCutter. *Parallel Computing*, 2001. To appear.
- [10] M. D. Beynon, T. Kurc, U. Catalyurek, A. Sussman, and J. Saltz. Efficient manipulation of large datasets on heterogeneous storage systems. In *Proceedings of the 11th Heterogeneous Computing Workshop (HCW2002)*. IEEE Computer Society Press, Apr. 2002.
- [11] M. D. Beynon, T. Kurc, A. Sussman, and J. Saltz. Optimizing execution of component-based applications using group instances. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid2001)*, Brisbane, Australia, May 2001.
- [12] Biomedical Informatics Research Network (BIRN). <http://www.nbirn.net>.
- [13] P. Brezany, A. Choudhary, and M. Dang. Parallelization of irregular codes including out-of-core data and index arrays. In *Proceedings of PARCO'97*, 1997.
- [14] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Parallel processing of spatial joins using R-trees. In *Proceedings of the 1996 International Conference on Data Engineering*, pages 258–265, Feb. 1996.
- [15] P. H. Carns, W. B. Ligon, R. B. Ross, and R. Thakur. PVFS: A parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, Oct 2000.
- [16] H. Casanova and J. Dongarra. Applying Netsolve's network-enabled server. *IEEE Computational Science & Engineering*, 5(3):57–67, July-September 1998.
- [17] Common Component Architecture Forum. <http://www.cca-forum.org>.

- [18] E. Cerami. *Web Services Essentials*. O'Reilly & Associates Inc., 2002.
- [19] C. Chang, R. Ferreira, A. Sussman, and J. Saltz. Infrastructure for building parallel database systems for multi-dimensional data. In *Proceedings of the Second Merged IPPS/SPDP Symposiums*. IEEE Computer Society Press, Apr. 1999.
- [20] P. Corbett, D. Feitelson, S. Fineberg, Y. Hsu, B. Nitzberg, J.-P. Prost, M. Snir, B. Traversat, and P. Wong. Overview of the MPI-IO parallel I/O interface. In *Proceedings of the IPPS'95 Third Annual Workshop on Input/Output in Parallel and Distributed Systems (IOPADS)*, pages 1–15, Apr. 1995.
- [21] P. F. Corbett and D. G. Feitelson. The Vesta parallel file system. *ACM Transactions on Computer Systems*, 14(3):225–264, Aug. 1996.
- [22] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing, 2001.
- [23] Data Access and Integration Services. [//http://www.cs.man.ac.uk/grid-db/documents.html](http://www.cs.man.ac.uk/grid-db/documents.html).
- [24] R. Das, Y.-S. Hwang, M. Uysal, J. Saltz, and A. Sussman. Applying the CHAOS/PARTI library to irregular problems in computational chemistry and computational aerodynamics. In *Proceedings of the 1993 Scalable Parallel Libraries Conference*, pages 45–56. IEEE Computer Society Press, Oct. 1993.
- [25] R. Das, M. Uysal, J. Saltz, and Y.-S. Hwang. Communication optimizations for irregular scientific computations on distributed memory architectures. *Journal of Parallel and Distributed Computing*, 22(3):462–479, Sept. 1994.
- [26] The DataCutter project. <http://www.cs.umd.edu/projects/hpsl/ResearchAreas/DataCutter.htm>.
- [27] D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. *Communications of the ACM*, 35(6):85–98, June 1992.
- [28] Distributed Oceanographic Data System. <http://www.unidata.ucar.edu/packages/dods/>.
- [29] G. Edjlali, A. Sussman, and J. Saltz. Interoperability of data parallel runtime libraries with Meta-Chaos. Technical Report CS-TR-3633 and UMIACS-TR-96-30, University of Maryland, Department of Computer Science and UMIACS, May 1996.
- [30] G. Edjlali, A. Sussman, and J. Saltz. Interoperability of data parallel runtime libraries. In *Proceedings of the 11th International Parallel Processing Symposium*, Geneva, Switzerland, Apr. 1997. IEEE Computer Society Press.
- [31] Earth Systems Grid (ESG). <http://www.earthsystemgrid.org>.
- [32] EUROGRID. <http://www.eurogrid.org/>.
- [33] C. Faloutsos and P. Bhagwat. Declustering using fractals. In *the 2nd International Conference on Parallel and Distributed Information Systems*, pages 18–25, San Diego, CA, Jan. 1993.
- [34] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In *the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Philadelphia, PA, Mar. 1989.
- [35] R. Ferreira, T. Kurc, M. Beynon, C. Chang, A. Sussman, and J. Saltz. Object-relational queries into multi-dimensional databases with the active data repository. *Parallel Processing Letters*, 9(2):173–195, 1999.
- [36] S. Fink, S. Kohn, and S. Baden. Efficient run-time support for irregular block-structured applications. *Journal of Parallel and Distributed Computing*, 50(1):61–82, Apr. 1998.
- [37] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid services for distributed system integration. *IEEE Computer*, 36(6):37–46, June 2002. Open Grid Services Architecture (OGSA).
- [38] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The physiology of the Grid: An open grid services architecture for distributed systems integration. <http://www.globus.org/research/papers/ogsa.pdf>, 2002.
- [39] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*. IEEE Press, Aug 2001.

- [40] Global Grid Forum. <http://www.gridforum.org>.
- [41] The Globus Project. <http://www.globus.org>.
- [42] S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. SAMS Publishing, 2002.
- [43] A. S. Grimshaw, W. A. Wulf, and the Legion Team. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, Jan. 1997.
- [44] Grid Physics Network (GriPhyN). <http://www.griphyn.org>.
- [45] E. Gutiérrez, O. Plata, and E. Zapata. A compiler method for the parallel execution of irregular reductions in scalable shared memory multiprocessors. In *Proceedings of the 14th ACM International Conference on Supercomputing*, pages 78–87, Santa Fe, New Mexico, May 2000.
- [46] M. Hall, S. Amarasinghe, B. Murphy, S. Liao, and M. Lam. Detecting coarse-grain parallelism using an interprocedural parallelizing compiler. In *Proceedings of Supercomputing '95*, San Diego, CA, Dec. 1995.
- [47] H. Han and C.-W. Tseng. Improving compiler and run-time support for adaptive irregular codes. In *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques*, Paris, France, Oct. 1998.
- [48] High Performance Fortran Forum. High Performance Fortran language specification. *Scientific Programming*, 2(1–2):1–170, 1993.
- [49] S. Hiranandani, K. Kennedy, and C.-W. Tseng. Compiling Fortran D for MIMD distributed-memory machines. *Communications of the ACM*, 35(8):66–80, Aug. 1992.
- [50] E. G. Hoel and H. Samet. Data-parallel spatial join algorithms. In J. Chandra, editor, *Proceedings of the 23rd International Conference on Parallel Processing. Volume 3: Algorithms and Applications*, pages 227–234, Boca Raton, FL, USA, Aug. 1994. CRC Press.
- [51] J. Huber, C. L. Elford, D. A. Reed, A. A. Chien, and D. S. Blumenthal. PPFs: A high performance portable parallel file system. In *Proceedings of the 9th ACM International Conference on Supercomputing*, pages 385–394, Barcelona, Spain, July 1995.
- [52] D. Kotz. Disk-directed I/O for MIMD multiprocessors. In *Proceedings of the 1994 Symposium on Operating Systems Design and Implementation*, pages 61–74. ACM Press, Nov. 1994.
- [53] T. Kurc, M. Beynon, A. Sussman, and J. Saltz. DataCutter and a client interface for the Storage Resource Broker with DataCutter services. Technical Report CS-TR-4133 and UMIACS-TR-2000-26, University of Maryland, Department of Computer Science and UMIACS, May 2000.
- [54] D.-R. Liu and S. Shekhar. A similarity graph-based approach to declustering problems and its applications towards parallelizing grid files. In *Proceedings of the International Conference on Data Engineering*, pages 373–381, Taipei, Taiwan, Mar. 1995. IEEE Computer Society Press.
- [55] Logistical Computing and Internetworking Lab. <http://www.loci.cs.utk.edu/>.
- [56] MEDIGRID. <http://creatis-www.insa-lyon.fr/MEDIGRID/home.html>.
- [57] B. Moon, H. Jagadish, C. Faloutsos, and J. Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, January/February 2001.
- [58] B. Moon and J. H. Saltz. Scalability analysis of declustering methods for multidimensional range queries. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):310–327, March/April 1998.
- [59] S. Mukherjee, S. Sharma, M. Hill, J. Larus, A. Rogers, and J. Saltz. Efficient support for irregular applications on distributed-memory machines. In *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP)*, pages 68–79. ACM Press, July 1995. ACM SIGPLAN Notices, Vol. 30, No. 8.
- [60] M. C. Murphay and D. Rotem. Multiprocessor join scheduling. *IEEE Transactions on Knowledge and Data Engineering*, 5(2):322–338, Apr. 1993.

- [61] N. Nieuwejaar and D. Kotz. The Galley parallel file system. In *Proceedings of the 1996 International Conference on Supercomputing*, pages 374–381. ACM Press, May 1996.
- [62] R. Oldfield and D. Kotz. Armada: A parallel file system for computational. In *Proceedings of CCGrid2001: IEEE International Symposium on Cluster Computing and the Grid*, Brisbane, Australia, May 2001. IEEE Computer Society Press.
- [63] R. Ponnusamy, J. Saltz, A. Choudhary, Y.-S. Hwang, and G. Fox. Runtime support and compilation methods for user-specified irregular data distributions. *IEEE Transactions on Parallel and Distributed Systems*, 6(8):815–831, Aug. 1995.
- [64] E. Port, M. Knopp, and G. Brix. Dynamic contrast-enhanced mri using gd-dtpa: Interindividual variability of the arterial input function and consequences for the assessment of kinetics in tumors. *Magn. Reson. Med.*, 45(6):1030–1038, 2001.
- [65] J.-P. Prost, R. Treumann, R. Hedges, B. Jia, and A. Koniges. MPI-IO/GPFS, an optimized implementation of MPI-IO on top of GPFS. In *Proceedings of the 2001 ACM/IEEE SC01 Conference*. ACM Press, Nov. 2001.
- [66] V. Raman, I. Narang, C. Crone, L. Haas, S. Malaika, T. Mukai, D. Wolfson, and C. Baru. Data access and management services on grid. <http://www.cs.man.ac.uk/grid-db/documents.html>.
- [67] M. Ranganathan, A. Acharya, G. Edjlali, A. Sussman, and J. Saltz. Flexible and efficient coupling of data parallel programs. In *Proceedings of Parallel Object-Oriented Methods and Applications (POOMA96)*, Feb. 1996.
- [68] J. M. d. Rosario and A. N. Choudhary. High-performance I/O for massively parallel computers: Problems and prospects. *IEEE Computer*, 27(3):59–68, Mar. 1994.
- [69] J. Saltz, G. Agrawal, C. Chang, R. Das, G. Edjlali, P. Havlak, Y.-S. Hwang, B. Moon, R. Ponnusamy, S. Sharma, A. Sussman, and M. Uysal. *Advances in Computers*, volume 45, chapter Programming Irregular Applications: Runtime Support, Compilation and Tools, pages 105–153. Academic Press, 1997.
- [70] J. Saltz, R. Ponnusamy, S. D. Sharma, B. Moon, Y.-S. Hwang, M. Uysal, and R. Das. A manual for the CHAOS runtime library. Technical Report CS-TR-3437 and UMIACS-TR-95-34, University of Maryland, Department of Computer Science and UMIACS, Mar. 1995.
- [71] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: a network based information library for a global world-wide computing infrastructure. In *Proceedings of HPCN'97 (LNCS-1225)*, pages 491–502, 1997.
- [72] D. Schneider and D. DeWitt. Tradeoffs in processing complex join queries via hashing in multiprocessor database machines. In *Proceedings of the 16th VLDB Conference*, pages 469–480, Melbourne, Australia, Aug. 1990.
- [73] K. E. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett. Server-directed collective I/O in Panda. In *Proceedings of Supercomputing '95*, San Diego, CA, Dec. 1995. IEEE Computer Society Press.
- [74] X. Shen and A. Choudhary. DPFS: A distributed parallel file system. In *IEEE 30th International Conference on Parallel Processing (ICPP)*, Sept. 2001.
- [75] X. Shen and A. Choudhary. A distributed multi-storage i/o system for high performance data intensive computing. In *International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, May 2002.
- [76] J. Smith, A. Gounaris, P. Watson, N. W. Paton, A. A. Fernandes, and R. Sakellariou. Distributed query processing on the grid. <http://www.cs.man.ac.uk/grid-db/documents.html>.
- [77] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI: The Complete Reference*. Scientific and Engineering Computation Series. MIT Press, 1996.
- [78] Shared Pathology Informatics Network (SPIN). <http://www.sharedpath.org>.
- [79] SRB: The Storage Resource Broker. <http://www.npaci.edu/DICE/SRB/index.html>.
- [80] M. Stonebraker and P. Brown. *Object-Relational DBMSs, Tracking the Next Great Wave*. Morgan Kaufman Publishers, Inc., 1998.

- [81] TeraGrid. <http://www.teragrid.org>.
- [82] D. Thain, J. Basney, S. Son, and M. Livny. Kangaroo approach to data movement on the grid. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*, 2001.
- [83] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Gathering at the well: Creating communities for grid i/o. In *Proceedings of Supercomputing 2001*, Denver, CO, November 2001.
- [84] R. Thakur, A. Choudhary, R. Bordawekar, S. More, and S. Kuditipudi. Passion: Optimized I/O for parallel applications. *IEEE Computer*, 29(6):70–78, June 1996.
- [85] M. Uysal, T. M. Kurc, A. Sussman, and J. Saltz. A performance prediction framework for data intensive applications on large scale parallel machines. In *Proceedings of the Fourth Workshop on Languages, Compilers and Run-time Systems for Scalable Computers*, pages 243–258. Springer-Verlag, May 1998.
- [86] S. Vazhkudai, S. Tuecke, and I. Foster. Replica selection in the globus data grid, 2001.
- [87] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, 1999.
- [88] H. Yu and L. Rauchwerger. Adaptive reduction parallelization techniques. In *Proceedings of the 14th ACM International Conference on Supercomputing*, pages 66–77, Santa Fe, New Mexico, May 2000.