# XML Database Support for Distributed Execution of Data-intensive Scientific Workflows*

Shannon Hastings†, Matheus Ribeiro‡, Stephen Langella†, Scott Oster†, Umit Catalyurek†, Tony Pan†, Kun Huang†, Renato Ferreira‡, Joel Saltz†, Tahsin Kurc†

† Dept. of Biomedical Informatics
The Ohio State University
Columbus, OH, 43210

‡ Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, MG - Brazil

## Abstract

*In this paper we look at the application of XML data management support in scientific data analysis workflows. We describe a software infrastructure that aims to address issues associated with metadata management, data storage and management, and execution of data analysis workflows on distributed storage and compute platforms. This system couples a distributed, filter-stream based dataflow engine with a distributed XML-based data and metadata management system. We present experimental results from a biomedical image analysis use case that involves processing of digitized microscopy images for feature segmentation.*

## 1 Introduction

The main purpose of data collection is to better understand the problem at hand and predict, explain, and extrapolate potential solutions and outcomes. On one hand, advances in computational and data acquisition technologies improved the resolution and speed at which a researcher can collect data. On the other hand, because of the increasing complexity and size of scientific datasets, data analysis is increasingly becoming a major challenge in research. Scientific data analysis involves several tasks, including 1) querying, retrieval, and integration of data of interest from large and distributed datasets, 2) simple and complex operations on data, and 3) inspection and visualization of results. By composing individual tasks into data analysis workflows, a researcher can create and execute several (potentially new) analysis paths efficiently. This can lead to a better insight to the problem, which may not be possible by processing of data by a single task only. A complex scientific workflow can consist of multiple stages of tasks organized into networks of operations and hierarchical structures (i.e., a task may itself be another workflow).

This work examines the effective application of distributed XML database support in scientific data analysis workflows. XML has become a de facto standard for data exchange in Web and Grid environments. A large number of tools have been developed for creating, parsing, validating, and querying XML documents. With an XML-aware approach, it becomes possible for both clients and application developers to leverage these tools. This paper makes the following contributions: 1) We identify a list of functionality that is desired for scientific workflows and can benefit from XML database support. 2) We describe a software framework that implements the desired functionality by coupling a distributed filter-stream based data processing middleware with a distributed XML data management middleware. The salient features of this system include support for managing data types, which are input to and output from a workflow component, as XML schemas, support for management of workflow descriptions, support for distributed execution of workflows, and on-demand database creation to store and retrieve output datasets and intermediate data products. We present an experimental evaluation of the system in an image analysis use case that involves processing of large biomedical images for feature segmentation.

## 2 Related Work

A number of research projects have developed tools and runtime infrastructure to support composition and execution of scientific workflows. Because of space constraints, we briefly review some of the related work. A good list and survey of workflow systems can be found at http://www.extreme.indiana.edu/swf-survey/.

The Chimera [4] system implements support for estab-

*This research was supported in part by the National Science Foundation under Grants #ACI-9619020 (UC Subcontract #10152408), #EIA-0121177, #ACI-0203846, #ACI-0130437, #ANI-0330612, #ACI-9982087, #CCF-0342615, #CNS-0406386, #CNS-0426241, Lawrence Livermore National Laboratory under Grant #B517095 (UC Subcontract #10184497), NIH NIBIB BISTI #P20EB000591, Ohio Board of Regents BRTTC #BRTT02-0003. Contact Author: Tahsin Kurc, kurc@bmi.osu.edu.

lishing virtual catalogs that can be used to describe and manage information on how a data product in an application has been derived from other data. This information can be queried, and data transformation operations can be executed to regenerate the data product. The Pegasus project develops systems to support mapping and execution of complex workflows in a Grid environment [3]. The Pegasus framework uses the Chimera system for abstract workflow description and Condor DAGMan and schedulers [5] for workflow execution. It allows construction of abstract workflows and mappings from abstract workflows to concrete workflows that are executed in the Grid. The Kepler project [10] develops a scientific workflow management system based on the notion of actors. Application components can be expressed as Actors that interact with each other through channels. The actor-oriented approach allows the application developer to reuse components and compose workflows in a hierarchical model. Adapting Computational Data Streams is a framework that addresses construction and adaptation of computational data streams in an application [8]. A computational object performs filtering and data manipulation, and data streams characterize data flow from data servers or from running simulations to the clients of the application. Dv [1] is a framework based on the notion of *active frames*. An active frame is an application-level mobile object that contains application data, called *frame data*, and a *frame program* that processes the data. Active frames are executed by *active frame servers* running on the machines at the client and at remote sites.

Our work differs from these projects in that we focus on management of metadata associated with workflows (e.g., definition of a workflow), input/output datasets, and data types exchanged between workflow components using a generic, XML-based metadata and data management system. The system presented in this paper allows storage and retrieval of data and workflow definitions as XML documents conforming to well-defined schemas and enables use of common protocols for storing and querying these documents. Our system could be used by other systems in order to store workflow definitions and instance data. Similarly, our system could use, for example, Condor [5] and Pegasus [3] for scheduling of computations in a Grid environment.

Moreau et. al. [11] propose the use of XML schemas to describe the logical structure of a scientific dataset. The mapping of the logical structure to the physical layout of the dataset is done through mapping documents. The goal is to provide a conceptual view of the dataset, with which applications and workflows can interact without worrying about the physical data format. Their approach is similar to ours in that they use XML schemas to define data types. However, our system, specifically the underlying Mobius framework [7, 9], also provides support for coordinated management and versioning of schemas, on-demand XML database creation, database federation, and querying in a distributed environment. Unlike the XDTM system in [11], which creates a mapping from an XML schema to physical format by mapping documents, the current implementation of Mobius manages XML documents in an XML database, called MakoDB [7, 9], layered on a relational database system. For a given XML schema, MakoDB automatically creates database tables and a mapping of the schema to these tables for efficient storage and querying of XML documents conforming to the schema.

## 3 Desired Functionality

In order to support complex scientific workflows, a workflow system should address a wide range of requirements. These requirements include user interfaces and languages for easy composition of workflows, workflow scheduling and execution, monitoring of workflows, management of datasets, reliability and robustness, and unified access to sources in the environment, to name a few. Most data analysis applications can be expressed as a network of data processing components that exchange data and control information to execute in a coordinated way. In this paper, we focus on data and metadata centric requirements associated with such workflows.

**Strongly typed data.** In a distributed and collaborative environment, a data analysis workflow may be composed of components developed by multiple, potentially independent researchers. In such an environment, metadata and data definitions play an important role. Metadata for workflows and definitions of data structures (data objects) exchanged among workflow components can provide a mechanism to ensure a workflow is composed correctly. Moreover, having metadata associated with workflows makes it easier to store, manage, and search for instances of workflows. Web and Grid services standards aim to address the related issues of interoperability between services and components. Nevertheless, support for management of data definitions (i.e., the structure of data objects and datasets, which are input to and output from workflow components) promotes data to be a first class citizen. One advantage of this data centric approach is that strongly typed data provides an additional mechanism to check if two interacting components in a workflow are compatible. Another advantage is, even if the data objects produced and consumed by two components are different, a user or application developer can inspect the structure of the data objects and implement transformation components that will map or convert one data object to the other.

**On-demand creation and management of distributed databases.** A workflow can be executed by having all its components run concurrently and exchange data on the fly across wide- or local-area networks. Another approach

would be to use a storage system to act as a *persistent data channel* between components at different stages of the workflow. Here, we use the term persistent data channel in the sense that a workflow component's output data object that is stored in the system can be used by multiple invocations of another component in the workflow or a component in another workflow. Such an approach allows more flexible scheduling of stages of a workflow onto available resources. If strongly typed data objects are persisted in the environment, they can be shared, searched, and queried. This not only enables persistent data channels, but also allows other clients and programs to interact with those data objects. By examining the intermediate datasets, a collaborator can better understand how the researcher arrived at her results. Furthermore, intermediate datasets could be utilized as input to a new workflow that builds on the original workflow. This could potentially provide a significant runtime performance improvement as redundant workflow steps do not need to be recomputed if they are shared between multiple workflows [12]. In order to support this type of functionality, the system should be able to create a database of data objects on demand and make it available for remote access. The system should take advantage of storage clusters for large storage space and high I/O performance.

**Management of data analysis workflows.** Researchers should be able to compose, register, and version workflows. The definitions and instances of workflows should be managed in a standard and efficient way so that researchers can share workflows and reference others' workflows in theirs. The system should also allow a researcher to version workflows (e.g., to add new analysis components or modify the order of data processing steps) and manage the updated instances of workflows.

**Efficient execution of data analysis workflows.** To speed up complex operations on data and parameter studies, the system should support execution of analysis workflows and manage flow of data within the network of components in a heterogeneous and distributed environment. It should also allow check-pointing of intermediate results so that the workflow can be restarted after a failure or the user can carry out interactive inspection on the data at a later time.

# 4 System Architecture

We have developed a software infrastructure that implements the functionality presented in Section 3 by coupling an XML-based distributed data and metadata management system, Mobius [7], with a distributed dataflow middleware, DataCutter [2].

## 4.1 Mobius and DataCutter

Mobius is a middleware framework designed for efficient metadata and data management in dynamic, distributed environments. It provides a set of generic services and protocols to support 1) creation, management, and versioning of XML schemas, 2) on-demand creation and federation of databases conforming to the XML schemas managed by the system, and 3) querying of these databases in a distributed environment. Its services employ XML schemas to represent metadata definitions and XML documents to represent and exchange metadata instances.

The *Mobius Global Model Exchange* (GME) service provides a protocol for publishing, versioning, and discovering XML schemas. It is implemented as an architecture similar to Domain Name Server (DNS), in which there are multiple GMEs each of which is an authority for a set of namespaces. A schema is stored in GME under a name and namespace specified by the application developer and is given a version number. We refer to the tuple consisting of the schema's name, its namespace, and its version number as the global name id (GNI) of the schema. The GME provides support for a schema to reference entities already existing in other schemas and in the global schema defined by a researcher. Other services such as storage services can use the GME to match instance data with their data type definitions. The *Mobius Mako* service exposes data sources as XML data services through a set of well defined interfaces and provides support for storing, updating, retrieving, and querying data as XML documents. The runtime support allows user-defined data types (represented as XML schemas) to automatically manifest custom databases at runtime, and data adhering to these data types to be stored in these databases. Any data instance received by a Mako server is validated against an XML schema as retrieved from the GME server and its elements are indexed. A Mako server can be configured to accept only a specific set of XML schemas. Clients and other services may query XML documents within a Mako by sending it an XPath statement.

DataCutter [2] implements a coarse-grained dataflow system using a filter-stream model. A DataCutter application consists of application-specific components, called *filters*, and one or more *filter groups*. Filters exchange data through a *stream* abstraction, which denotes uni-directional flow of data from one filter to another. The application filters read buffers from their input ports, perform application-defined processing on the buffer, and then write it to their output ports. The DataCutter runtime allows combined use of task- and data-parallelism. Filters can be placed on different machines and multiple copies of a filter can be created and executed across heterogeneous collections of storage and compute nodes.

## 4.2 Implementation

In our system, the input and output data types of a component in a workflow can be described by data schemas. A data schema is an application specific entity and can describe any user-defined structure and attributes that can

be encoded as an XML schema. For workflows, we have adapted the process modeling schema (PMS) developed for the Distributed Process Management System (DPM) [6]. It defines a hierarchical model starting with a *workflow* entity which contains several *jobs*. Each job represents an application which is composed of a set of *components*. The name attribute of a component allows the component to be named such that it can be uniquely identified. Each component entity also contains input and output list attribute. An optional placement attribute specifies how many copies of a component should be run in the environment and which nodes to run the copies of the component. The PMS and data schemas are registered in and managed by the Mobius GME service.

Our system uses the Mako service of Mobius as the backbone for storage and management of data instances. We utilize a collection of Makos distributed across storage nodes to provide a distributed data management service. Workflow instance descriptions, metadata describing the datasets, and data instances are stored in Makos. An instance of the application workflow is modeled by a directed acyclic task graph of components represented by an XML document conforming to the PMS. The instance specifies the function names and locations of individual components, the number of copies and placement of copies for a component, persistent check-pointing locations in the workflow (which tells the execution environment that output from check-pointed components should be stored as intermediate datasets in the system), input and output datasets (conforming to some registered schema), and an optional data selection criteria (which specifies the subset of data elements from input datasets to be processed). Using Mako client APIs, clients can search for a particular workflow and execute it using the distributed execution service.

The distributed execution service builds on DataCutter and is responsible for instantiation of components on distributed platforms, management of data flow between components, and data retrieval from and storage to distributed collections of Makos. We have implemented two filters, *MakoReader* and *MakoWriter*, that provide interface between Mako servers and other filters in the workflow. The *MakoReader* filter retrieves data from Mako servers and passes them to the first stage filters in the workflow. The *MakoWriter* filter can be placed between two application filters, which are connected to each other in the workflow graph, if the output of a filter needs to be check-pointed. The last stage filters in the workflow also connect to *MakoWriter* filters to store output in Mako servers. To maximize I/O parallelism when data is accessed, the system utilizes data distribution techniques for storing data through the MakoWriter filter. Currently, round-robin and demand-driven (based on the data ingestion rate of individual Makos) strategies are implemented for data distribution. The execution of a workflow and check-pointing can be done stage-by-stage (i.e., all the data is processed by one stage and stored on Mako servers before the next stage is executed) or in pipelined fashion (i.e., all stages execute concurrently; the *MakoWriter* filters interspersed between stages both send data to Mako servers and pass it to the next filter in the workflow). In addition to MakoWriter and MakoReader filters, any workflow component can utilize Mako client APIs to store XML documents and can retrieve data using XPath expressions.

## 5 Experimental Evaluation

We performed an evaluation of our framework using PC clusters and an image analysis use case. The first set of experiments examines the performance of the system as the number of Mako servers that can store data is varied. A cluster with 7 nodes was used for this experiment. Each node of the cluster consists of two AMD Opteron processors with 8 GB of memory, 1.5 TB of disk storage in RAID 5 SATA disk arrays. The nodes are inter-connected via a Gigabit switch. The workflow consists of a simple chain of *MakoReader->Inverter->MakoWriter* filter group with 7500 images as input – each image was a 256x256-pixel gray scale image. The Inverter filter inverts the color of each pixel in an image. In the experiments, the number of Inverter filter copies was fixed at 7 and each copy was executed on one of the nodes. The input images were distributed evenly across an increasing number of Mako servers as the number of MakoReader filters increased. The results of these experiments are shown in Figure 1. The numbers in the graphs are the total execution time for processing 7500 images. The bars labeled as "Reader" and "Writer" show the execution times when the number of MakoReader and MakoWriter filters is varied, respectively. We observe that the execution time decreases as the number of Mako servers is increased. As more Makos are added, better I/O parallelism is achieved.

In the next set of experiments, we use a biomedical image analysis application implemented as a chain of data processing operations, as shown in Figure 2. This application performs segmentation of the labyrinth layer in a digitized microscopy image of a mouse placenta[1]. The implementation consists of two main stages. The RedFinder, Counter, and Histogram Aggregation filters form the first processing stage of the application and process data in filter-stream fashion. The second stage consists of a single filter referred to here as the PCA filter. Data exchange between the two stages is done through the XML database support. The output of the Counter filter is stored in Mobius and retrieved by the PCA filter. We now briefly describe these stages.

[1]The segmented region can be used to carry out quantitative examination of phenotypes and measurements of tissue structure within the placenta in cancer research.

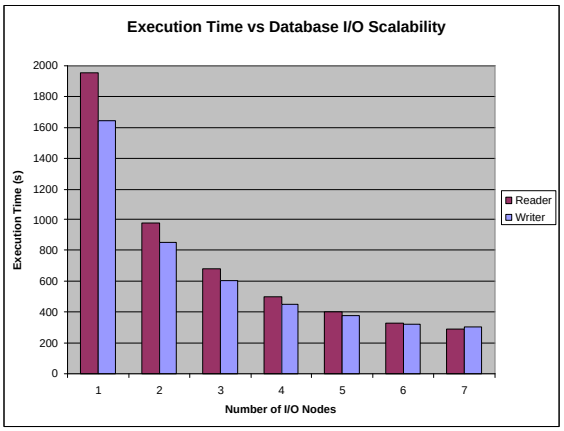


**Figure 1. Data I/O Scalability Experiments. "Reader" and "Writer" bars denote the experiments, in which the number of MakoReader and MakoWriter filters is varied, respectively.**

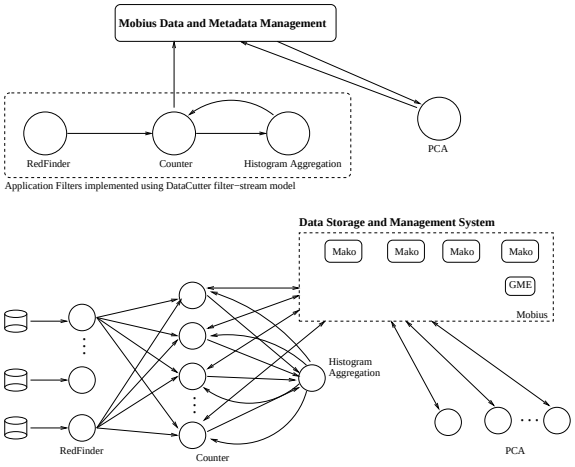


**Figure 2.** Top: The network of filters which form the processing skeleton of the application. Bottom: An instance of the application with multiple copies of RedFinder, Counter, and PCA filters.

**RedFinder, Counter, Histogram Aggregation:** These filters find red pixels in regions with high red blood cell (RBC) density. The RedFinder filter reads image chunks, each of which represents a rectangular subregion of the image – in the experiments, input images are stored in files in the system – and determines if a pixel is red or not. The coordinates of red pixels are sent to the Counter filter, which counts the number of neighbors of each red pixel. A histogram of red pixel density is generated by the Histogram Aggregation filter. Using the histogram, the regions with high RBC density (specified by a user-defined threshold value) are determined. The Counter filter finds the red pixels whose coordinates fall into the high density regions, groups them into chunks, creates an XML document for each chunk, and sends using the Mobius client API the XML documents along with the chunk to Makos for storage. The schema for the XML document consists of attributes that specify the encoding type of the image (JPEG, TIFF, raw RGB, etc.), the name (or id) of the image, the id and bounding box of the chunk, and the size of the chunk. The chunk is submitted as an attachment to the XML document.

**PCA:** This filter retrieves the chunks stored by the Counter filter in the system and processes them to: 1) Compute the principal direction of the high RBC density regions. We apply principal component analysis (PCA) to the high RBC density regions to determine the orientation of these regions. 2) Determine the bounding box for the labyrinth layer by projecting pixels along the principle direction and filtering out those pixels whose projected coordinates fall outside a user-defined range. Multiple copies of the PCA filter can be instantiated to process an image in parallel. During execution, every filter copy first submits an XPath query, which specifies the id of the image, to Mobius and retrieves the list of chunks that satisfy the query. This list is partitioned evenly among filter copies in round-robin fashion. Each filter copy then retrieves the chunks assigned to itself from Mobius for processing using XPath queries. The output from the second stage conforms to the same schema that defines the output of the Counter filter and is stored in Mobius.

We carried out an experimental evaluation of the application implementation on a PC cluster, referred to here as *OSUMED*. Each node has a Pentium III 900MHz CPU, 512MB main memory, and three local disks (300GB local storage space). The nodes in the cluster are inter-connected through a 100Mbit/sec Ethernet Switch. We used digitized microscopy images, each of which was 40GB in size. The results represent the execution time for a single image averaged over 3 images. The amount of data stored in Mobius by the Counter and PCA filters was equal to 1.4GB and 800MB per image, respectively.

In Figure 3, the execution time of the Counter filter is broken down into the operations performed by that filter; doing neighborhood computations and writing data to Mobius. In these experiments, we executed four Mako instances. The round-robin distribution strategy was employed when inserting data to backend Mako instances. As is seen from the figure, the overhead of storing data in Mobius is very small compared to the neighborhood computations. The overhead of neighborhood computations decreases as the number of filter copies is increased as expected. The cost of storage in Mobius remains almost constant as the number of backend Makos was fixed at 4 in these experiments. Our results show that interaction with the XML-based storage services do not become a bottleneck in this configuration as the number of filters writing data is increased. The last set of experiments shows the execution

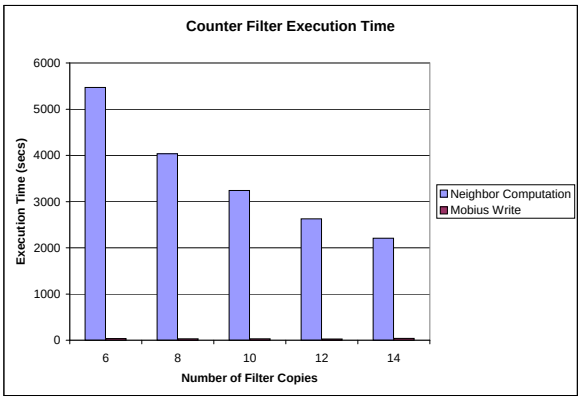


**Figure 3. The breakdown of execution of the Counter filter.**

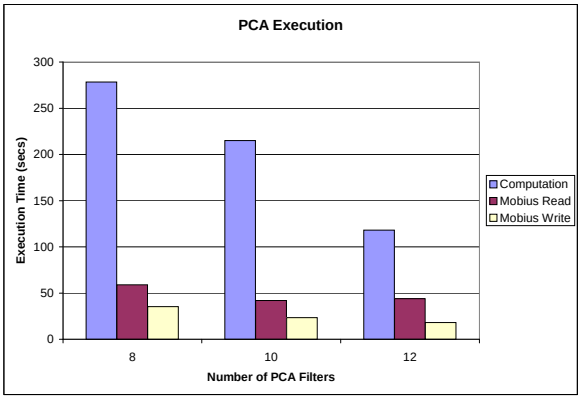


**Figure 4. The execution time of the PCA filter as the number of filter copies is varied. The figure also shows the time spent reading data from Mobius and writing data to Mobius.**

time of the second stage of the image analysis application (the PCA computations). In our implementation, the second stage is executed after the first stage is completed, i.e., after the redFinder-Counter-Histogram Aggregation filter group has finished its execution. As is seen from Figure 4, the PCA execution time decreases as more filters are executed. The I/O overhead (Mobius Read/Write) is less than the PCA execution time for this experimental setup as well. However, since the number of Mobius Mako servers is fixed, the cost of I/O remains almost constant.

## 6 Conclusions

This paper examined how XML database support can be applied in the context of scientific workflows. We argue that XML is a viable technology for management and execution of workflows. To demonstrate our ideas, we presented and evaluated a system that uses a distributed XML-based metadata/data management system in tandem with a component-based distributed execution engine.

## References


[1] M. Aeschlimann, P. Dinda, J. Lopez, B. Lowekamp, L. Kallivokas, and D. O'Hallaron. Preliminary report on the design of a framework for distributed visualization. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 1833–1839, Las Vegas, NV, June 1999.

[2] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed processing of very large datasets with DataCutter. *Parallel Computing*, 27(11):1457–1478, Oct. 2001.

[3] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1), 2003.

[4] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, 2002.

[5] 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.

[6] S. Hastings. Distributed architectures: A java-based process management system. Master's thesis, Computer Science Department, Rensselear Polytechnic Institute, 2002.

[7] S. Hastings, S. Langella, S. Oster, and J. Saltz. Distributed data management and integration: The mobius project. In *GGF Semantic Grid Workshop 2004*, pages 20–38. GGF, June 2004.

[8] C. Isert and K. Schwan. ACDS: Adapting computational data streams for high performance. In *14th International Parallel & Distributed Processing Symposium (IPDPS 2000)*, pages 641–646, Cancun, Mexico, May 2000.

[9] S. Langella, S. Hastings, S. Oster, T. Kurc, U. Catalyurek, and J. Saltz. A distributed data management middleware for data-driven application systems. In *Proceedings of 2004 IEEE International Conference on Cluster Computing*, September 2004.

[10] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows, to appear*, 2005.

[11] L. Moreau, Y. Zhao, I. Foster, J. Voeckler, and M. Wilde. XDTM: the XML Dataset Typing and Mapping for Specifying Datasets. In *Proceedings of the 2005 European Grid Conference (EGC'05)*, Amsterdam, Netherlands, Feb. 2005.

[12] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Pipeline and batch sharing in grid workloads. In *Proceedings of High-Performance Distributed Computing (HPDC-12)*, pages 152–161, Seattle, Washington, June 2003.