The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration

Ian Foster[1,2] Jens Vöckler[2] Michael Wilde[1] Yong Zhao[2]
[1] Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA
[2] Department of Computer Science, University of Chicago, Chicago, IL 60637, USA
{foster,wilde}@mcs.anl.gov, {voeckler,yongzh}@cs.uchicago.edu

Abstract

It is now common to encounter communities engaged in the collaborative analysis and transformation of large quantities of data over extended time periods. We argue that these communities require a scalable system for managing, tracing, communicating, and exploring the derivation and analysis of diverse data objects. Such a system could bring significant productivity increases, facilitating discovery, understanding, assessment, and sharing of both data and transformation resources, as well as the productive use of distributed resources for computation, storage, and collaboration. We define a model and architecture for a *virtual data grid* to address this requirement. Using a broadly applicable "typed dataset" as the unit of derivation tracking, we introduce simple constructs for describing how datasets are derived from transformations and from other datasets. We also define mechanisms for integrating with, and adapting to, existing data management systems and transformation and analysis tools, as well as Grid mechanisms for distributed resource management and computation planning. We report on successful application results obtained with a prototype system called Chimera that implements some of these concepts, involving challenging analyses of high-energy physics and astronomy data.

1 Introduction

Much interesting research in data systems is concerned, directly or indirectly, with facilitating the extraction of insight from large quantities of data. This problem has motivated innovative techniques for translating data into accessible and interpretable forms (relational databases, metadata, curation), for dealing with large quantities of data and complex queries (database organization, query optimisation), and for applying databases to various classes of problem.

We propose here a more expansive view of data system architecture based on an integrated treatment of not only data but also the computational procedures used to manipulate data and the computations that apply those procedures to data. This integrated treatment is motivated by two observations: first, in many communities, programs and computations are significant resources—sometimes even more important than data; and second, understanding the relationships among these diverse entities is often vital to the execution and management of user and community activities. We call the result of this integration a *virtual data system*, because (among other things) it allows for the representation and manipulation of data that does not exist, being defined only by computational procedures.

In such a virtual data system, data, procedures, and computations are all first class entities, and can be published, discovered, and manipulated. Thus we can, for example, trace the provenance of derived data; plan and track the computations required to derive a particular data product; determine whether a requested computation has been performed previously and whether it is cheaper to rerun it or to retrieve previously generated data; and discover computational procedures with desired characteristics. As we continue to experiment with virtual data concepts and implementations, we continue to be surprised by new applications.

Our work is motivated and informed by (1) the requirements of communities of physical scientists with whom we work within the GriPhyN project, in domains such as high energy physics and astronomy, and (2) our experience developing a prototype virtual data system, Chimera [11], that is undergoing preliminary evaluation within several such communities [1]. In this article, we draw upon this experience both to motivate our approach

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment

Proceedings of the 2003 CIDR Conference

and to permit preliminary evaluation of the effectiveness of our techniques. But we also reach significantly beyond this initial prototype to address issues that arise when creating a virtual data grid (VDG) capable of encompassing the diverse expertise of large multidisciplinary communities in a scalable fashion.

The practical realization of the VDG concept introduces a variety of challenging technical problems associated with the need to integrate diverse resources (data, computational procedures, storage systems, computers) along multiple dimensions (discovery, access, authorization, provisioning, etc.) and at multiple levels of abstraction (files, relations, datasets, virtual data, etc.). We address these challenges by defining (1) general but powerful abstractions for representing data and computation, (2) a virtual data schema capable of representing key entities and relationships, and (3) a VDG architecture that builds specialized techniques for representing and maintaining virtual data information on top of an Open Grid Services Architecture (OGSA) foundation [9].

Our proposed architecture touches upon issues of provenance and federation that have been extensively studied in the data systems community (see Section 7). However, this article goes beyond previous work in three respects. First, it makes the case for a new approach to data system design that embraces a wide spectrum of data generation and representation modalities. Second, it presents a comprehensive (although necessarily high-level) design for a system that realizes this approach. Third, it presents preliminary large-scale evaluations of the approach in challenging application scenarios.

The rest of this article is as follows. In Section 2, we summarize our current understanding of the problem that we seek to solve. In Section 3 we introduce our virtual data model. In Section 4, we introduce the key elements of the VDG architecture. In Section 5, we talk about how we envision the VDG being used. In Section 6, we describe the capabilities of our Chimera prototype and present results from early application experiments. We discuss related work in Section 7, future directions in Section 8, and conclude in Section 9.

2 The Virtual Data Problem

We assume a (potentially large) number of both *datasets* and *procedures*. Datasets live in storage systems and are accessible over the network via service interfaces. Procedures may live in network-accessible storage systems and/or be invocable as services. Datasets and procedures may be curated to varying extents, meaning that they are subject to validation, documentation, versioning, etc., and in general have a quality vouched for by various authorities. All entities may be geographically distributed, be subject to different access control policies, and have varying performance characteristics. Users work individually or collaboratively to invoke procedures, which may extract information from datasets, update datasets, and/or create or delete datasets.

2.1 Critical Tasks

We observe that the following tasks can be difficult and could be facilitated via appropriate tools:

Discovery: locating datasets, transformations, and computations of interest and value, based on the information and knowledge accumulated in virtual data catalogs. (E.g., "I want to search an astronomical database for galaxies with certain characteristics. If a program that performs this analysis exists, I won't have to write one from scratch." "I want to apply an astronomical analysis program to millions of objects. If the program has already been run and the results stored, I'll save weeks of computation.")

Documentation: annotating datasets and procedures with user-defined and -supplied metadata.

Provenance: determining the validity of data by accessing an audit trail describing how the data was produced from the datasets and previous data derivations on which it depends. ("I've discovered some interesting data, but need to understand the corrections applied when it was constructed before I can trust it for my purposes." "I've detected a calibration error in an instrument and want to know which derived data to recompute.")

Sharing: of both data and transformation resources, facilitating the productive use of distributed grid resources for computation, storage and collaboration.

Productivity: is enhanced by increased sharing and powerful discovery techniques that make it easier to learn, both through metadata and example, what data are available within a community, and how they can be used.

2.2 Goals

Our goal is an integrated, scalable management solution for a distributed collection of datasets, procedures, and computer resources. Such a system must address, derivation, discovery, planning, and estimation:

- *Derivation*: Track relationships among derived datasets and the procedures that generate them, in order to support subsequent querying/navigation of these relationships.
- *Discovery*: Locate, and determine how to access, a dataset or procedure with specified attributes. This is a conventional metadata search, with the added wrinkles that attributes of interest may refer to derivation relationships.
- *Planning*: Allocate resources (computers, storage, networks) in response to requests for data products and procedure invocations. Examples include replication of popular datasets and procedures, and reclamation of resources of lesser value.
- *Estimation*: Determine the cost of executing a procedure. This information can be vital input to both provisioning and user query planning decisions.

It may seem that this problem statement introduces unnecessary complexity by mingling apparently independent issues. However, there are significant benefits to treating these issues in an integrated fashion. For example, resource requirements recorded with provenance information can be used to guide subsequent planning decisions, while the identity of the physical resources used for a particular derivation may be relevant to subsequent provenance tracking if two executions of the same program do not generate identical results.

3 Virtual Data Schema

The system that we have developed to address these requirements comprises two primary components: a *virtual data schema* that defines the objects to be maintained and manipulated within the VDG, and relationships among these objects; and a *virtual data system* that allows an individual or community to construct and maintain this information in a distributed, decentralized context. We describe the schema here and the virtual data system in the next section.

The virtual data schema defines the data objects and relationships of the virtual data model. An implementation within a particular virtual data service instance might be a relational database, object-oriented database, XML repository, or even a hierarchical directory such as a file system or LDAP database.

Our virtual data schema defines six classes of objects (Figure 1). The *dataset* and *replica* objects capture information about datasets, while the *transformation* object captures information about the procedure construct. Both datasets and dataset-valued transformation arguments are designated as being of a *type*. The *derivation* and *invocation* objects capture information about specific instances of a transformation and, in so doing, also capture provenance relationships. The objects in Figure 1 capture a provenance relationship between a dataset foo of type "type2" produced by applying a transformation prog1 to a dataset fnn.

In more detail, a *dataset* is the unit of data managed within our virtual data model. Each dataset has a *type*, which specifies various characteristics of the dataset, including how it is structured or represented on storage or data servers and what kind of data it contains; *replica* allows for datasets with copies at multiple locations [6].

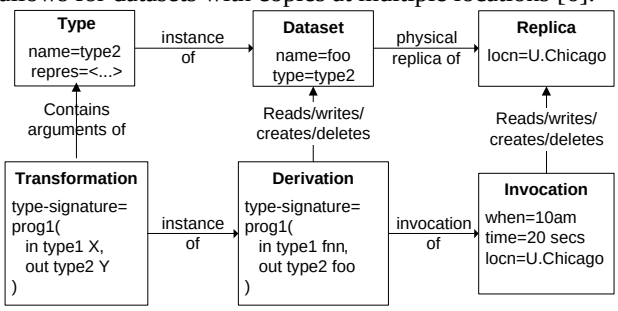


Figure 1: Major objects of the virtual data schema.

A *transformation* is a typed computational procedure that may take as arguments both strings passed by value and datasets passed by reference. A transformation may create, delete, read, and/or write datasets passed as arguments. We distinguish between a *simple* transformation, which acts as a black box, and a *compound* transformation, which composes one or more transformations in a directed acyclic execution graph.

A *derivation* specializes a transformation by specifying the actual arguments (strings and/or datasets) and other information (e.g., in some situations, environment variable values) required to perform a specific execution of its associated transformation. A derivation record can serve both as a historical record of what was done and as a recipe for operations that can be performed in the future. In the former case, it may record a provenance relationship between one or more datasets and a transformation. For example, the derivation object depicted in Figure 1 captures the fact that dataset foo was produced by applying transformation prog1 to dataset fnn.

An *invocation* specializes a derivation by specifying a specific environment and context (e.g., date, time, processor, OS) in which its associated derivation was executed. Specific dataset replicas can be associated with a particular invocation for tracking and diagnostic purposes, to keep a detailed account of provenance in an environment where datasets can be replicated.

Our data model specifies for each object a set of required attributes while also allowing for arbitrary additional attributes to capture application-specific information. We now describe each object in turn.

3.1 Datasets

The data model that underlies our virtual data system introduces a *dataset* abstraction to allow for the tracking of data in more general forms than "files," "tables," etc., and to insulate users from low-level data representations.

A dataset is the unit of data manipulated by a transformation. A primary purpose of the virtual data system is to track dataset provenance. (We adopt "dataset" as a generic term without reference to its many prior meanings and implementations.) A dataset is a unit of data that may be stored in any of a variety of containers. Our model includes *replicas* as a means of tracking multiple invocations of a derivation and the resulting datasets.

Depending on community and application, the datasets manipulated by a transformation might be variously:

- A single file or a set of files that are viewed as a single logical entity.
- A list of files with an associated offset-length pair specifying data to be extracted from each file.
- A set of files in a tar archive or some other archive format.

- An index file and a set of data files: for example, a gdbm database or a set of rows to be extracted, by primary key, from a SQL database.
- A closure of object references from a persistent object database.
- A region of a spreadsheet.

For each dataset the virtual data system maintains a *descriptor* which tells a transformation how the dataset is mapped onto a storage service. A dataset's descriptor provides the information needed by a transformation to access and manipulate the dataset's contents.

For example, if the dataset's contents are located in a single file, then the descriptor can be simply a file name. If the contents are a slice of a set of files, then the descriptor will provide both a list of file names and slice indices. If the dataset's contents are a set of rows in a database, then the descriptor will name a database and specify those rows, and so on. We do not define a fixed schema for describing dataset representations: a particular collaboration or user must define a set of descriptor schemas that are interpretable by its transformations.

3.2 Types

We introduce a *type model* to facilitate the discovery of transformations and datasets, enable error checking of derivation specifications, and guide optimisation.

A dataset has one or more type attributes: exactly one *representational* attribute and any number of additional *logical* attributes. The value of the representational attribute specifies the format of the dataset's *type descriptor*, which contains information about how the dataset is formatted. Type attributes can be thought of as distinguished metadata attributes that (a) define the representation of datasets and (b) determine what datasets can validly be passed as arguments to a transformation. Type attributes can be defined hierarchically, thus allowing for specialization. A leaf (sub)type attribute is referred to as concrete; other (parent) type attributes are referred to as abstract.

The following rules govern the use of types within the virtual data system.

1. Any dataset, dataset-value transformation argument, or dataset-value derivation argument can have multiple type attributes, from independent type attribute hierarchies, including, as noted above, exactly one representational type attribute, and any number of logical types attributes.
2. Any dataset type attribute must be concrete (i.e., a leaf node in its type attribute hierarchy); dataset-value transformation and derivation arguments can have either abstract or concrete attribute types.
3. Type conformance of datasets to transformation arguments is defined as follows: when a dataset is used as a derivation argument, each of its type attributes must be a concrete subtype of exactly one type attribute of the associated transformation argument.

We use the two type attribute hierarchies below to illustrate these concepts. A dataset might be defined as having type attributes MultiFileSet *and* MonteCarloSimulation, with the first attribute indicating the dataset's representation and the second its semantic content. However, a dataset cannot be defined as being of the abstract type attribute "EventCollection," whereas a transformation argument can.

FileDataset (abstract, representational)
  File (concrete, representational)
  FileSet (abstract, representational)
    MultiFileSet (concrete, representational)
    TarFileSet (concrete, representational)

EventCollection (abstract, logical)
  RawEventSet (concrete,logical)
  SimulatedEventSet (abstract, logical)
    MonteCarloSimulation (concrete, logical)
    DiscreteSimulation (concrete, logical)

We note that this type model differs from that of programming languages and databases in important ways. It does not describe the detailed contents of files in the manner that an abstract data type defines the fields of an object, nor does it, in a strict sense, define the operations that can be performed on a dataset. Its main purpose is to support flexible representations of datasets, discovery of datasets and transformations, and type checking of derivations. It does, however, employ the concepts of sub-typing and multiple inheritance from the type models of programming languages.

3.3 Transformations

We believe that a virtual data system must shield its users from low-level details of how data and procedures are represented, so that they can focus on higher-level questions of how data is produced and transformed. Just as the *dataset* provides a typed abstraction for arbitrary data containers, so the *transformation* provides a typed abstraction for arbitrary computational procedures.

We propose a general model for transformations that will (ultimately) be able to encompass the following:

- An executable for a particular architecture.
- A source program packaged to allow compilation and installation on a range of platforms.
- A script passed to an interpreter, such awk, perl, or python, or a command shell
- A set of SQL statements passed to a SQL query interpreter
- Commands for a general-purpose data manipulation package, such as SAS or SPSS.
- An application-specific package: e.g., in high energy physics, scripts in PAW or ROOT, or algorithms in ATHENA.

- A set of macros or an automation script passed to a visual application such as Excel.
- A Web service with interface defined by Web Services Description Language (WSDL).
- An invocation of a COM or COM+ ActiveX object.

A transformation type specification indicates for each transformation argument its directionality (IN or OUT) and its type, which may be either "string" or a dataset type as described above. Type signatures facilitate discovery, automated checking of interfaces, and eventually, execution plan optimisation.

Transformations that receive their arguments and input files via parameter files can defined as two-stage transformations, where the first stage takes VDL parameters and places them into a text file, and the second stage invokes the actual executable, passing it the text file produced by the first stage. Such couplings can conveniently be expressed using the "compound transformation" construct described in [11].

An important issue not yet addressed in our design is the structured versioning of transformations and mechanisms for managing compatibility assertions among different versions. It is important that we be able not only to track precisely what version of a transformation was executed to derive a given dataset, but also to express "equivalence" among different versions.

4 Virtual Data System Architecture

Having defined a virtual data schema, we turn to the question of how to maintain and provide access to that information in a distributed, multi-user, multi-institutional environment, to address our larger goals of scalability, manageability, and support for discovery and sharing.

We introduce the term virtual data catalog (VDC) to denote a service that maintains information defined by our virtual data schema. A VDC is, in general, an abstract notion: while we can imagine a single database that maintains a coherent, authoritative view of all known datasets, transformations, derivations, and invocations, the creation of such a database is rarely likely to be feasible in our assumed distributed, multi-user, multi-institutional environment. Instead, VDC contents will typically be distributed over multiple information resources with varying degrees of authenticity and coherency. Thus, in the following we first discuss issues raised by location and organization, then describe our approach to establishing authenticity, and finally outline the infrastructure elements used to support these mechanisms.

4.1 VDC Distribution and Integration

A community's virtual data catalog information may be distributed across multiple information repositories in a variety of ways and for a variety of reasons, including ownership and curation responsibilities (e.g., archives owned by different groups or individuals), the need to integrate with information resources maintained for other purposes (e.g., a metadata catalog or source code archive), replication for performance purposes, and a desire by subgroups or individuals to maintain independent "overlay" information that enhances information maintained by other groups.

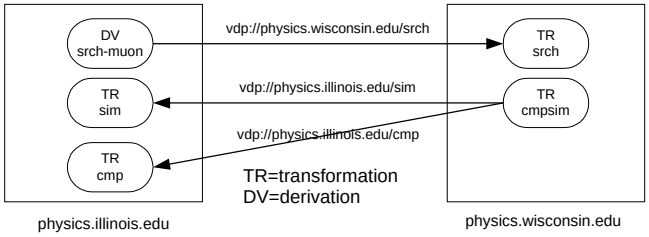


Figure 2: Virtual data hyperlinks between servers

Figure 2 depicts a scenario in which transformation and derivation records are distributed across two sites. (The corresponding dataset records are not depicted.) In this model of a distributed high-energy physics collaboration, the Wisconsin group is able to define a compound transformation "cmpsim" composed of two transformations created and maintained by a remote group working in llinois. The first stage of the transformation, "sim", performs a simulation operation, while "cmp" compresses the result in a domain-specific manner. In turn, the Illinois group defines a derivation "srch-muon" that specifies the parameters needed to invoke the Wisconsin particle-searching application "srch" for the particular particle class "muon."

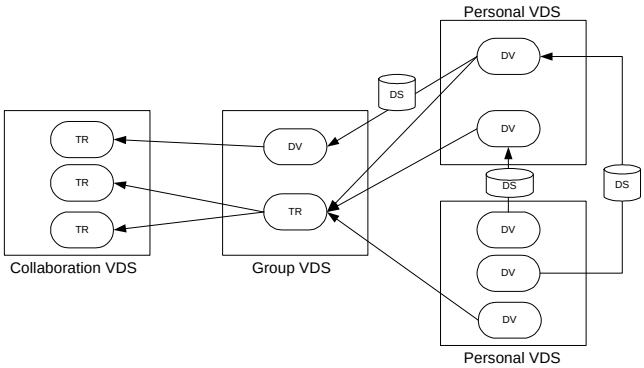


Figure 3: Provenance hyperlinks between virtual data servers

This example shows how distributed collaboration is facilitated by the capability of inter-catalog references. When standardized in the manner of uniform resource locators, such information descriptors can, we feel, revolutionize the way that knowledge-intensive distributed collaborations are conducted over large distances and scales. Such knowledge-enriched hyperlinks need not be limited to references to remote type and transformation records. Derivation provenance chains can also span across servers, as illustrated in Figure 3, with for example group derivations depending on derivations

in a collaboration-wide catalog and personal derivations depending on those of colleagues. As this type of technology becomes ubiquitous within and between collaborations in various disciplines, it will become possible for a researcher to click on a graph or table in a scientific paper, and discover in great detail and with high precision exactly how that dataset was produced.

Hyperlinked provenance information facilitates the integration and federation of VDC information contained in multiple catalogs. Given the wide variety of information sources, information qualities, and application needs that may be encountered in a virtual data grid, we can expect a variety of integration/federation approaches to be useful, ranging from central servers to federated databases, Google-like systems, and peer-to-peer structures. Some possibilities are illustrated in Figure 4. Four catalogs ("VDCs") maintained at different locations for different purposes and with different scopes provide direct local access to their contents. In addition, a variety of federated indexes integrate information about selected objects from multiple such catalogs. Presumably such federating indexes would be differentiated according to their scope (user interest, all community data, community approved data, etc.), accuracy (depth of index, update frequency), cost, access control, and so forth.

More generally, we envision that in an effective collaborative process, data and knowledge definitions will propagate across, up, and around the web of each virtual organization's knowledge servers as information is created, reprocessed, annotated, validated, and approved for broader use, trust, and distribution.

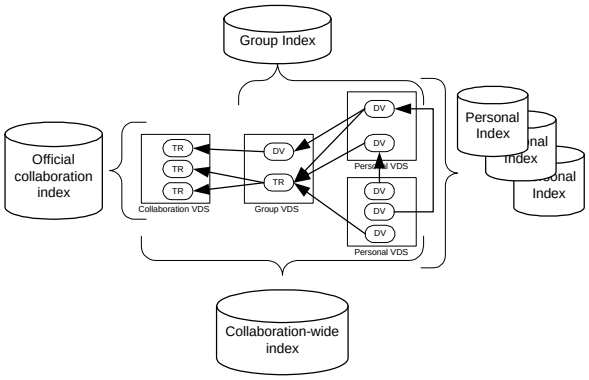


Figure 4: Indexing the VDG at multiple levels

4.2 Quality and Security

An important aspect of VDC community process is the maintenance of information concerning the "quality" of VDC entries. We use this generic term to indicate various quantitative and qualitative measures that a community or individuals may apply concerning such issues as curation, authorship, authenticity, and timeliness. Some such measures are tied strongly to process: for example, in a highly curated collection, each transformation, dataset, and derivation chain might be assessed, audited, and approved according to defined procedures. In other cases, "quality" might correspond to an annotation applied by a computational procedure, while some users might apply more ad hoc measures, for example trusting data produced by certain individuals.

As is the case with other aspects of VDG design, our goal is to establish basic machinery that can be used to implement a wide variety of approaches and policies. In this context, we note that the distributed, multi-user, multi-institutional nature of the VDG environments means that, in general, we (a) must introduce automated and secure techniques for verifying trust and (b) cannot rely on direct trust relationships among individuals. Thus, we choose to use cryptographic signatures on VDC entries and attributes as a means of establishing the identity of the authority(s) that vouch for their validity. When embedded in a framework that provides for establishing root authority(s) and for validating trust chains, these mechanisms can be used to implement a wide variety of security models and policies. Similar mechanisms can be used for access control, as the policies enforced by a resource "owner" are likely to require similar recourse to authority.

4.3 Infrastructure

The realization of the concepts described above requires a variety of enabling infrastructure, including mechanisms for establishing inter-catalog references (and, in general, for naming VDG entities); establishing identity and authority; service discovery; virtualizing compute resources; and so forth. We do not discuss these issues here except to note that our current prototype builds on Globus Toolkit v2 technology and that our intention is to build future systems on the Grid infrastructure defined within OGSA [9] and implemented by the Globus Toolkit v3. OGSA and its Web services (WS) foundation together address issues of naming, service discovery, service characterization, notification, authentication and authorization (via the Grid Security Infrastructure [10], Community Authorization Service [17], and WS security, perhaps with extensions), and service provisioning and management, among other critical issues.

An infrastructure component that is vital to our ability to perform dynamic resource provisioning is an effective *resource virtualization* facility. Ideally, such a facility would allow an arbitrary hardware resource to be configured to meet the needs of an arbitrary transformation; the required configuration would then form part of the description of the transformation, and a scheduler could take the cost of achieving this configuration into account when selecting resources. One approach to realizing this goal is to use hosting environment technologies such as J2EE and .NET. However, complex scientific applications also introduce native compiled code, multiprocessor execution, and other requirements that conventional hosting environments are

not equipped to deal with. In this context, other approaches to virtualization can be appropriate, such as the Condor remote execution facility (system calls are trapped and returned to the originating site) and the Globus Toolkit's Grid Resource Allocation and Management (GRAM) protocol, which allows, for example, for application-specific environment variable settings, prestaging of input data, redirection of standard output, and poststaging of output data.

5 Application Context and Benefits

We have described our schema for representing virtual data, and the components and architecture of the VDG. We can now address our central thesis: namely, that these constructs can indeed be of benefit to particular data-intensive user communities. We explain how the virtual data mechanisms outlined above can be integrated with (and of benefit to) typical scientific and technical computing workflows. In particular, we show how our model ties in to the six key facets of the VDG process flow: *composition*, *planning*, *derivation*, *estimation*, *discovery*, and *sharing* (see Figure 5) focusing on questions of how derivation data is captured, discovered, used, and managed and discarded.

We start by assuming that we are working within a community context within which:

- The collaboration has carefully crafted its processing paradigm and created and defined a set of transformations that form the basic toolkit of most application scientists and production engineers.
- Users can extend that toolkit by creating new transformations, often composing them from existing transformations.
- Mechanisms to automatically track transformations are integrated into interactive systems.
- Output datasets produced by the execution of derivations are automatically marked.

Within this framework, we describe each of these processes in the sections that follow.

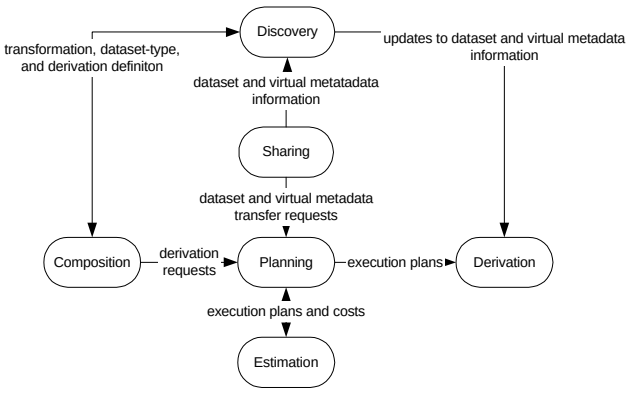


Figure 5: Virtual data process flow

5.1 Composition

We use the term composition to refer to the entire process of creating virtual data definitions for all of the objects that make up the virtual data schema: dataset-types, datasets, transformations, derivations, and invocations. We envision that this activity will be integrated in both manual and automated manners.

Production managers—those in a collaboration responsible for planning large, structured, official computations, often taking place over months or years—will carefully structure a space of derivations and submit requests from that "virtual data" space. Individual researchers will manually create definitions as needed for smaller data spaces, and typically request the derivation of that data shortly after it has been defined. Both user groups will use VDL to specify these data spaces.

In addition to such "batch" scenarios, we envision VDL being integrated into interactive analysis tools so that researchers exploring data spaces in a less structured fashion have the benefits of a historical log of their recent data derivation activities. These users can then choose to snapshot these logs (which could be maintained directly in a VDC) into a more permanent and well-categorized and named portion of their virtual data workspace. Over time, as these definitions accumulate, they become significant personal and community resources.

5.2 Planning Data Access and Computation

Once derivations are defined in the VDC, users (and automated production mechanisms) can request that these virtual datasets be "materialized." We term the process of mapping these requests to the Grid "planning," as it is suggestive of the database query planning process.

Grid request planning is a challenging research area that involves tracking the state of both request queues and grid resources (networks, computing elements, and storage systems), and being cognizant of the complex and potentially overlapping resource usage policies set by both physical and virtual organizations within the Grid. The planner must allocate resources (computers, storage, networks) in response to requests for data products and procedure invocations, and make decisions to replicate popular datasets and procedures either on demand and/or via pre-staging [19].

The application of procedures to datasets can be performed in a variety of ways, including the following.

1. *Procedure collocated with data*. A dataset may be accessible via a service interface that supports specific operations.
2. *Ship procedure to data*. Alternatively, a dataset may be accessible via a service interface that allows for the execution of user-specified procedures: for example, SQL queries or arbitrary executables. A user may construct such procedures on the fly, or retrieve them from another source.

3. *Ship data to procedure*. A procedure may be accessible via a service interface that allows for the upload of datasets to which the procedure is then applied. A user may construct such datasets on the fly, or retrieve them from another source.
4. *Ship procedure and data to computer*. In an environment in which workloads exceed the capacity of servers that host data (#1 or #2) or procedures (#1, #3), it can be useful to be able to integrate additional computational resources, for example by instantiating new copies of data or of services.

All four patterns can play a role in a particular community or application, depending on factors such as resource availability and performance, the size of datasets, and the computational and data demands of procedures.

5.3 Estimation

Planning requires that we be able to obtain an estimate of the cost of executing the data derivation workflow graph (both computation and data transfer nodes) for each candidate plan. This information can be vital input not only for automated request planning but also for user query planning: interactive users may query the estimator directly to assess whether or not a particular desired virtual data product can be computed in the time that the user is willing to wait for it.

5.4 Derivation

Derivation refers to the process of running transformations with specified arguments to produce specific datasets. Derivation is conducted by workflow management systems that dispatch computation and/or data transfer requests to specific grid sites and monitor their completion, dispatching nodes of the workflow graph when predecessor dependencies have completed. An example of such a scheduler is the Condor DAGman.

This process produces the invocation records in the virtual data schema that record details for each execution of a derivation—data, time, execution site, and execution environment (OS, processor type, host name, etc).

The automated planning and derivation of large and complex workflows can be an important productivity multiplier, permitting users to explore aspects of their data space that were previously inaccessible due to the large burden of planning and managing such computations. The promise is that tasks that were previously both computationally intense and exceedingly difficult to plan, execute, monitor, and correct, now become automated. The goal is that the Grid becomes like an enormously powerful workstation, and the virtual data catalog an invaluable source of recipes to run on that facility.

5.5 Discovery

Discovery is the process of locating, and determining how to access, a dataset or procedure with specified attributes. The capabilities of conventional metadata searches are enhanced in the VDG by the added possibilities that attributes of interest may refer to derivation relationships and that users may wish to search for data that may exist as "data" and/or in terms of recipes for generating that data. The dataset type, through its more precise characterization of semantics, representation, and interface, further enhances the precision of searching for data and procedure.

6 Experiences

The vision and system design presented in this paper are the outgrowth of our problem domain analysis and implementation and application experience within the GriPhyN project [2]. Within GriPhyN, we have implemented two generations of the Chimera Virtual Data System, and have applied it to challenge problems derived from the large-scale scientific collaborations that are collaborators in GriPhyN. We describe here these two implementations and survey the current application work in progress. More detail on the system design and implementation can be found in [11] and on one of the major application efforts in [1].

We used a first version, Chimera-0, to study the database schema needed to represent provenance relationships. This version (as well as its successor, Chimera-1) was aimed at representing transformations that consisted of single invocations of executable programs under a POSIX model of program execution. (The POSIX model implies an executable that resides in a file, which is passed arguments both on the command line and via named "environment variables", and which can access files through the open() system call.) The Chimera-0 schema consisted of a basic mapping of the POSIX execution semantics into a "transformation" object, with each invocation being a separate object.

Using this mechanism, we were able to create Chimera database definitions for a high energy physics collision event simulation application that consisted of four separate program executions with intermediate and final results passing between the stages as files. For the last two stages the files were in fact object-oriented database files from a commercial OODBMS product.

We learned from this effort what is needed to describe accurately applications with complex parameters and behaviour. We also created "canonical" applications that mimic arbitrary argument passing conventions and file I/O behaviour, and used these to create large application dependency graphs to validate our provenance tracking mechanism [11].

We have also addressed a larger challenge problem from astrophysics, namely the analysis of data from the Sloan Digital Sky Survey via the application of the MaxBCG galaxy cluster detection algorithm. This work [1] involved a much larger volume, a more realistic workload, and more complex data dependency tracking. We created and executed dependency graphs for

searching for galaxy clusters in the entire currently available survey, creating about 5000 derivations. We processed one third of the current survey data collection, using workflow DAGs with as many as several hundred executable nodes, across a grid consisting of almost 800 hosts spread across four sites, and using as many as 120 hosts in a single workflow.

We are currently implementing challenge problems involving more interactive analysis processing models than these large, batch-oriented challenges. For both ATLAS and CMS, we are prototyping environments in which we can track data produced in a set of multi-stage simulations, iterate in an unstructured manner over a small number of changeable analysis codes, select and filter interesting events, produce "cut sets" of events that meet certain physics properties, produce a series of histograms from the final analysis, and combine these cut sets into graphs that visualize interesting properties and relationships in the data. The data representations in these challenges include files, relational databases, and persistent object repositories, enabling us to test ideas for handling what we refer to as "multi-modal" data. Our goal is to be able to produce, for each data point in the final graph, a detailed data lineage report on the datasets that contributed to the creation of that point.

7 Related Work

The importance of documenting provenance is well known [18]. Our work builds on preliminary explorations within GriPhyN [3, 12]. There are also relationships to work in database systems [4, 5, 19] and versioning [15]. Cui and Widom [7, 8] record the relational queries used to construct materialized views in a data warehouse, and then exploit this information to explain lineage. Our work can leverage these techniques, but differs in two respects: first, data may not be stored in databases and the operations used to derive data items may be arbitrary computations; second, we address issues relating to the automated generation and scheduling of the computations required to instantiate data products.

Early work on *conceptual schemas* [12] introduced virtual attributes and classes, with a simple constrained model for the re-calculation of attributes in a relational context. Subsequent work produced an integrated system for scientific data management called ZOO [13], based on a special-purpose ODBMS that allowed for the definition of "derived" relationships between classes of objects. In ZOO, derivations can be generated automatically based on these relationships, using either ODBMS queries or external transformation programs. Chimera is more specifically oriented to capturing the transformations performed by external programs, and does not depend on a structured data storage paradigm or on fine-grained knowledge of individual objects that could be obtained only from an integrated ODBMS.

We can also draw parallels drawn between Chimera and workflow [14, 16] and knowledge management systems that allow for the definition, discovery, and execution of (computational) procedures.

Our thoughts on large-scale maintenance of community knowledge have similarities to, and are in part inspired by, Semantic Web concepts [3], although our application domain has unique characteristics.

We view as a significant open issue the question of how query optimisation techniques can be applied to planning issues that arise in VDGs. To date, work in this area has focused on lower-level scheduling issues relating for example to data movement [19].

8 Future Directions

Looking beyond the ideas and designs proposed here, we already envision both significant new capabilities that further elevate the level at which users interact with computing resources and practical extensions to make the proposed design more realistic and usable.

We envision that our future work will yield powerful general-purpose browsers that, within scientific and knowledge-intensive disciplines, are capable of making the discovery process as easy to use as today's Internet search engines, but with the added precision of formal queries on precisely specified interfaces.

The most significant new area that we intend to explore is that of extending our virtual information base into a knowledge base. Using knowledge representation, search, and inference techniques, we envision raising the level of interaction with the VDG to a domain-cognizant model in which searches and work requests are specified in the terminology and concepts of the domain(s) whose data, transformations, and knowledge are maintained in the VDS. The design proposed here will serve as a powerful base on which to conduct these explorations.

We plan other extensions designed to make our design more robust and usable. These include the following:

1) A model for tracking the provenance of datasets that reside in relational or object-oriented databases at a fine level of granularity. This is especially relevant in light of the case that is being made to have large collaborations keep all data assets, online, in a database.

2) A model for representing equivalence and similarity between data products. For example, two datasets created by the same derivation at different points in time may not be bitwise identical, but may be equivalent in their behaviour and semantics for a certain class of transformations.

We intend to explore the commonalities between code and data, and the similarity of our system for tracking data dependencies and those for tracking code (source) code and executable dependencies (e.g., "make"). An ideal system would integrate or even unify these concepts and mechanisms.

We also seek to explore a concept we call "virtual datasets," in which multiple datasets refer to different overlaid subsets of the same physical storage elements. This construct raises difficult issues of storage management and garbage collection.

Among the hard problems that need to be addressed and solved to bring this work to fruition are the following:

- Integrating provenance tracking mechanisms into existing tools, both general (such as SAS, SPSS, Excel, etc) and specialized, such as the ROOT and PAW analysis environments for high energy physics.
- Dealing with "update" as an operation a proc can perform on a DS; this maintains provenance but looses re-createability unless there is a transaction log for some type of undo operation.
- Implementing large shared catalog that can be accessed across an enterprise-scale collaboration, with scalability and availability

9 Conclusions

We have argued that at least in scientific and technical computing (and we suspect elsewhere), an increased focus on both data-intensive and collaborative approaches to problem solving leads to a need to manage the data, procedures, and computations performed by a community as an integrated and interrelated whole.

We have outlined the essential architectural elements of a *virtual data grid* system designed to address this requirement. This architecture defines a virtual data schema to represent the principal shared objects and the relationships among those objects, and a set of supporting mechanisms for managing the maintenance of this information. From a database perspective, VDGs not only represent a novel application domain but also introduce new technical problems in such areas as provenance. From a distributed systems perspective, VDGs have the attractive property of being able to leverage Grid systems in an interactive but disciplined fashion.

We have also described a prototype virtual data system, Chimera, and its application to scientific data analysis problems. Initial results with the Chimera prototype suggest that at least some of the benefits we claim for virtual data systems can be realized in practice.

Clearly this article only scratches the surface in terms of what it means to create and apply usable virtual data grids. We have defined what seems to be a workable architecture and demonstrated feasibility in real applications, but further study is required before we can determine whether or not such systems really do accelerate the problem solving process, and whether our architectural constructs can scale as required.

Acknowledgements

We gratefully acknowledge helpful discussions with Adam Arbree, Raj Bose, Peter Buneman, Rick Cavanaugh, Peter Couvares, Ewa Deelman, Catalin Dumitrescu, Greg Graham, Carl Kesselman, Miron Livny, Alain Roy, and our colleagues on GriPhyN, iVDGL and PPDG. This research was supported in part by the National Science Foundation under contract ITR-0086044 (GriPhyN), and by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38 (Data Grid Toolkit).

Appendix A – Chimera Virtual Data Language Version 1

We present a slightly simplified version of the core elements of the currently implemented Chimera virtual data language (VDL): those that allow us to represent the definition of transformations and their execution, and see how these enable the tracking of data derivation and provenance. We show the textual version of VDL here; an XML version is also implemented for machine-to-machine interfaces. A basic transformation looks like this:

```
TR t1( output a2, input a1, none env="100000",
none pa="500" )
{
  argument parg = "-p "${none:pa};
  argument farg = "-f "${input:a1};
  argument xarg = "-x -y ";
  argument stdout = ${output:a2};
  application = "/usr/bin/app3";
  profile env.MAXMEM = ${none:env};
}
```

Derivations represent the execution of a transformation with a specific set of arguments: in other words, a procedure invocation. A derivation of transformation t1 above might look like this:

```
DV d1->example1::t1(
    a2=@{output:"run1.exp15.T1932.summary"},
    a1=@{input:"run1.exp15.T1932.raw"},
    env="20000",
    pa="600" );
```

When a derivation uses as input the output of a previous derivation, a dependency graph is created. The VDL records the information necessary to capture this dependency. In the following example, file2, the output of trans1 produced by derivation usetrans1, is used as the input to trans2 in derivation usetrans2. This is the essence of data provenance tracking in Chimera.

```
TR trans1( output a2, input a1 )
{  argument stdin = ${input:a1};
   argument stdout = ${output:a2};
   application = "/usr/bin/app1";
}

TR trans2( output a2, input a1 )
{
  argument stdin = ${input:a1};
```

```
  argument stdout = ${output:a2};
  application = "/usr/bin/app2";
}

DV usetrans1->trans1( a2=@{output:"file2"},
a1=@{input:"file1"} );

DV usetrans2->trans2( a2=@{output:"file3"},
   a1=@{input:"file2"} );
```

Three simple transformations, and the fourth transformation, trans4, which is a compound transformation composed of calls to trans1, 2, and 3:

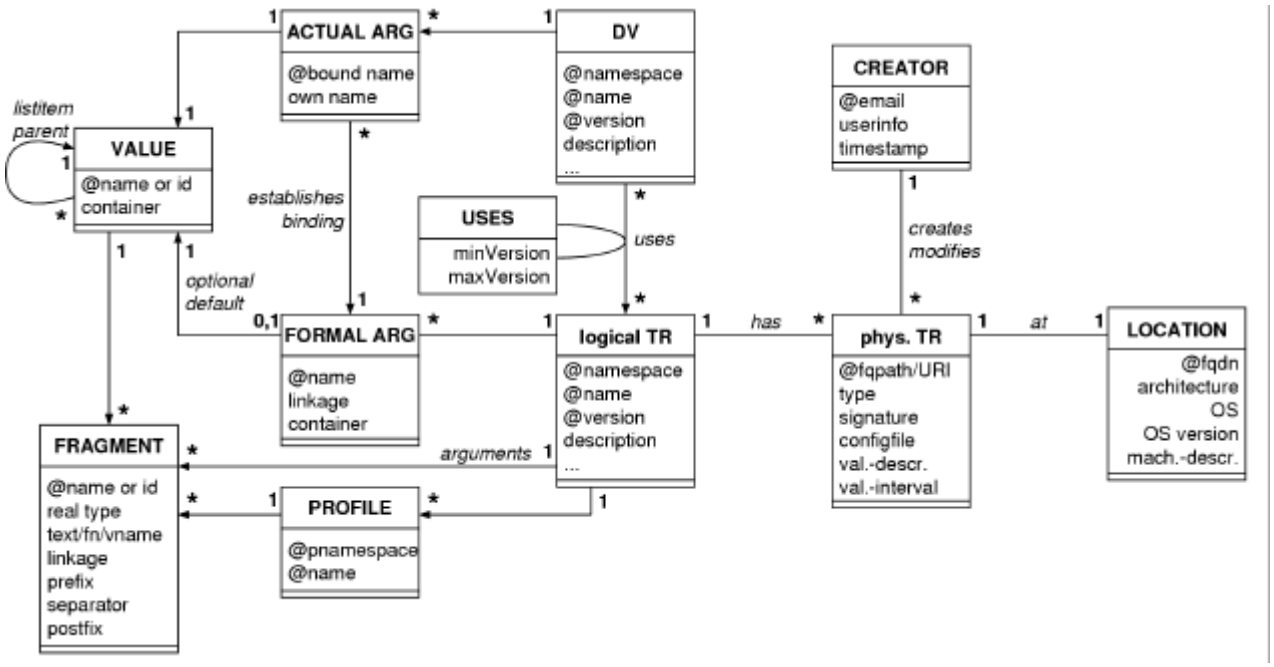
```
TR trans1( output a2, input a1 ) {
  argument = "...";
  argument stdin = ${input:a1};
  argument stdout = ${output:a2};
  application = "/usr/bin/app1";
}

TR trans2( output a2, input a1 ) {
  argument = "...";
  argument stdin = ${input:a1};
  argument stdout = ${output:a2};
  application = "/usr/bin/app2";
}

TR trans3( input a2, input a1, output a3 )
{
  argument parg = "-p foo";
  argument farg = "-f "${input:a1};
  argument xarg = "-x -y -o "${output:a3};
  argument stdin = ${input:a2};
  application = "/usr/bin/app3";
}
```

```
TR trans4(input a2,
          input a1,
          inout a5=@{inout:"anywhere":""},
          inout a4=@{inout:"somewhere":""},
          output a3 )
{
  call trans1( a2=${output:a4}, a1=${a1} );
  call trans2( a2=${output:a5}, a1=${a2} );
  call trans3( a2=${input:a5},
      a1=${input:a4}, a3=${output:a3} );
}
```

Another transformation, trans5, which is a compound transformation composed of the simple transformation trans1 and the compound transformation trans4:

```
TR trans5(input a2,
          input a1,
          inout a4=@{inout:"someplace":""},
          output a3 )
{
  call trans1( a2=${output:a4}, a1=${a1} );
  call trans4( a2=${input:a4}, a1=${a2},
               a3=${a3} );
}
```

Appendix B - Chimera Virtual Data Catalog Schema – VDL 1.0



REFERENCES

1. Annis, J., Zhao, Y., Voeckler, J., Wilde, M., Kent, S. and Foster, I., Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey. in *SC'2002*, (2002).
2. Avery, P. and Foster, I. The GriPhyN Project: Towards Petascale Virtual Data Grids, 2001. www.griphyn.org.
3. Berners-Lee, T., Hendler, J. and Lassila, O. The Semantic Web. *Scientific American*.
4. Buneman, P., Khanna, S., Tajima, K. and Tan, W.-C., Archiving Scientific Data. in *ACM SIGMOD International Conference on Management of Data*, (2002).
5. Buneman, P., Khanna, S. and Tan, W.-C., Why and Where: A Characterization of Data Provenance. in *International Conference on Database Theory*, (2001).
6. Chervenak, A., Deelman, E., Foster, I., Guy, L., Hoschek, W., Iamnitchi, A., Kesselman, C., Kunszt, P., Ripeanu, M., Schwartzkopf, B., Stockinger, H., Stockinger, K. and Tierney, B., Giggle: A Framework for Constructing Scalable Replica Location Services. in *SC'02*, (2002).
7. Cui, Y. and Widom, J., Practical Lineage Tracing in Data Warehouses. in *16th International Conference on Data Engineering*, (2000), 367–378.
8. Cui, Y., Widom, J. and Wiener, J.L. Tracing the Lineage of View Data in a Warehousing Environment. *ACM Transactions on Database Systems*, *25* (2). 179–227.
9. Foster, I., Kesselman, C., Nick, J. and Tuecke, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Globus Project, 2002. www.globus.org/research/papers/ogsa.pdf.
10. Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S. A Security Architecture for Computational Grids. in *ACM Conference on Computers and Security*, 1998, 83-91.
11. Foster, I., Voeckler, J., Wilde, M. and Zhao, Y., Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. in *14th Conference on Scientific and Statistical Database Management*, (2002).
12. Ioannidis, Y.E. and Livny, M. Conceptual Schemas: Multi-faceted Tools for Desktop Scientific Experiment Management. *International Journal of Cooperative Information Systems*, *1* (3). 451-474.
13. Ioannidis, Y.E., Livny, M., Gupta, S. and Ponnekanti, N., ZOO : A Desktop Experiment Management Environment. in *22th International Conference on Very Large Data Bases*, (1996), Morgan Kaufmann, 274-285.
14. Leymann, F. and Altenhuber, W. Managing Business Processes as an Information Resource. *IBM Systems Journal*, *33* (2). 326–348.
15. Marian, A., Abiteboul, S., Cobena, G. and Mignet, L., Change-Centric Management of Versions in an XML Warehouse. in *27th International Conference of Very Large Data Bases*, (2001).
16. Mohan, C., Alonso, G., Gunthor, R. and Kamath, M. Exotica: A Research Perspective on Workflow Management Systems. *Data Engineering Bulletin*, *18* (1). 19-26.
17. Pearlman, L., Welch, V., Foster, I., Kesselman, C. and Tuecke, S., A Community Authorization Service for Group Collaboration. in *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, (2002).
18. Williams, R., Bunn, J., Moore, R. and Pool, J. Interfaces to Scientific Data Archives, Center for Advanced Computing Research, California Institute of Technology, 1998.
19. Woodruff, A. and Stonebraker, M. Supporting Fine-Grained Data Lineage in a Database Visualization Environment, Computer Science Division, University of California Berkeley, 1997.