

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

Ian Foster^{1,2} Jens Vöckler² Michael Wilde¹ Yong Zhao²

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 increasingly common to encounter communities engaged in the collaborative analysis and transformation of large quantities of data over extended periods of time. We argue that these communities require a scalable system for managing, tracing, exploring and communicating 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 facilitating the productive use of distributed resources for computation, storage, and collaboration. Thus, we define a model and architecture for a *virtual data grid* capable of addressing this requirement. We define a broadly applicable model of a “typed dataset” as the unit of derivation tracking, and 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 implementation called Chimera, 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 easily accessible and interpretable forms (relational databases, metadata, curation), for dealing efficiently with large quantities of data and complex queries (database organization, query optimization), and for applying databases to various classes of problem.

In this paper we propose 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; discover computational procedures with desired characteristics; and so on. 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, that is undergoing preliminary evaluation within several of these communities. In this article, we draw upon this experience both to motivate our approach 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 virtual data grid architecture that builds specialized techniques for representing and maintaining virtual data information on top of an Open Grid Services Architecture (OGSA) foundation [9]. Our goal, we believe, is an integrated, scalable management solution for a distributed collection of datasets, procedures, and computer resources that addresses discovery, derivation, composition, estimation, and provisioning.

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 significantly 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 the results of preliminary but 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 virtual data grid architecture. In Section 5, we talk about the ways in which we envision the virtual data grid would be used. In Section 6, we describe the capabilities of our Chimera prototype, and present results from early application experiments with this prototype. 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 or, alternatively, be invocable as network-accessible 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.

Within this context, we observe that the following tasks tend to be hard, 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 gaining access to a complete audit trail describing how the data was produced from the datasets and previous data derivations on which it depends. (E.g., “I’ve come across some interesting data, but I need to understand the nature of 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: large resources available with less effort; plus the productivity that flows from increased sharing and enhanced discovery; easier to learn what data assets are available and how they can be used, both through metadata and through example.

The following are examples of the sorts of questions that we might want to ask.

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 issues:

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 of planning decisions 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, on the surface, 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, tracking provenance might appear to have little to do with planning of resource allocations; but 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 on different hardware systems.

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 principal 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 virtual data schema here, and the virtual data system in the next section.

The virtual data schema is so named because it defines the data objects and relationships of the virtual data model. Its implementation, with in instance of a virtual data service, may variously be a relational database, OO database, XML repository, or even a hierarchical directory such as a file system or LDAP database.

Our virtual data schema defines five classes of objects¹, as illustrated in Figure 1. The *dataset* and *replica* objects capture information about the dataset construct introduced above, while the *transformation* object captures information about the procedural construct. The *derivation* and *invocation* objects capture information about specific instances of a transformation and, in so doing, also capture provenance relationships. The example attribute values in the Transformation, Dataset, and Derivation objects capture a provenance relationship between a dataset foo produced by applying a transformation prog1 to a dataset fnn.

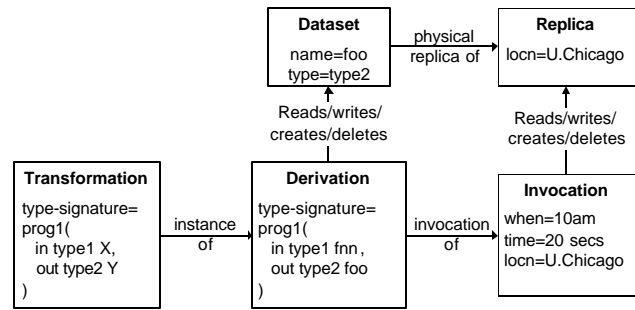


Figure 1: The five basic objects maintained within our virtual data schema.

In more detail:

A *dataset* is the unit of data managed within our virtual data model. Each dataset has a *dataset type*, which specifies various characteristics of the dataset, including how it is structured and what kind of data it contains.

The *replica* is introduced to allow for datasets that may have multiple physical copies with different properties such as location.

A *transformation* is a typed computational procedure that may take as arguments both strings, which are passed by value, and datasets, which are 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 also 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 replicas of datasets 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.

For each object, our data model specifies a set of required attributes while also allowing for the definition of arbitrary additional attributes used to capture application-specific information. In the subsections that follow, we describe each of these objects in turn.

¹ We could consider a dataset-type itself as a sixth class of object, but for the moment this would have little benefit for our model.

3.1 Data Model

The data model that underlies our virtual data system introduces the *dataset* abstraction to allow for the tracking of data in more general forms than “files,” “tables,” etc., and a *type model* to facilitate the discovery of transformations and datasets, and error checking in derivation specifications.

As we noted above, we want our virtual data system to shield its users from low-level details of data representation. To this end, we introduce an abstraction called a *dataset* to represent the unit of data manipulated by a program and/or whose derivation we wish to track. (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 wide variety of containers. In addition, as described in a later section, our model includes *dataset replicas* as a means of tracking multiple invocations of a derivation and the resulting data products.

A fundamental challenge faced by any general collaboration system for tracking provenance is the need to handle different data representations. Depending on community and application, the dataset manipulated by a particular transformation might be variously:

- A single file.
- 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.
- A set of rows to be extracted, by primary key, from one or more tables of a SQL database.
- A closure of object references from a persistent object database.
- A set of cell region references denoting a segment of a spreadsheet.

We propose to handle this wide spectrum of data representation through the following model:

A dataset definition maps a dataset *name* to a dataset *type* and a dataset *descriptor*.

A dataset’s *type* comprises three dimensions, which specify variously the dataset’s semantic *content*, the *format* of its physical representation, and the *encoding* used in that representation. Each component is represented simply as a string (e.g., “cms-simulation,” “text file”) with support for hierarchically defined subtypes (lists of strings) to allow for generalization and specialization. Note that as we are not working within the confines of any single programming language or system, there are no predefined “base types”: each Chimera user community can define their own set of type names, although a model in which some types become broad-

based standards can also be envisioned. An example of a dataset-type hierarchy is presented in Appendix C.

A dataset’s *descriptor* provides all information needed to access and manipulate the dataset’s contents. The nature of this descriptor will depend on the nature of the dataset. For example, if the dataset’s contents are located in a single file, then the descriptor can be a simple 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.

Each dimension of a fully specified dataset type can be specialized into subtypes. Within each dimension, types are arranged into a hierarchy of sub-types, to allow for generalization and specialization. The base types of the three type dimensions are defined as “Dataset-content,” “Dataset-format,” and “Dataset-encoding,” respectively, and we define “Dataset” as a synonym for the collective base-type <“Dataset-content”, “Dataset-format”, “Dataset-encoding”>. Thus, formal transformation arguments that are defined simply as “Dataset” are essentially untyped: any dataset can be supplied as an actual argument.

We should point out that while the Chimera dataset-type model derives some of its characteristics from the sophisticated type models of modern programming languages and database systems (e.g., Java, C++, Postgres), it also contains some notable differences and simplifications. In particular, it does not attempt to 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 discovery and type checking on transformation. It does, however, employ the concepts of sub-typing and multiple inheritance (in the form of dimensions) from the type models of programming languages.

3.2 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 Chimera *dataset* provides a generic, typed abstraction for arbitrary data containers, so the Chimera *transformation* provides a generic, 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, PAW, ROOT, or 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 is defined by a *transformation type* and a *transformation descriptor*. By documenting the types of the arguments operated on by a transformation, these type signatures facilitate discovery, automated checking of interfaces, and eventually, execution plan optimization.

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. Additionally, any transformation argument can be types as a list of dataset-types, indicating that the transformation can accept a union of types for that argument.

The general rule of type conformance in our virtual data model is simple: *a dataset can be supplied as an actual argument to a transformation if the type of the dataset is a proper subtype of the type list of the formal argument*. In other words, a dataset must be of a type that the transformation is equipped to process.

Transformations that expect to receive their arguments and input files via a parameter file are handled by defining them as two stage-transformations, where the first stage takes VDL parameters and places them into a text file, and the second stage which 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 next to the question of how to maintain and provide access to that information in a distributed, multi-user, multi-institutional environment, with a view to addressing 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. This 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 VDC information may be distributed across multiple information resources 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), need to integrate with information resources maintained for other purposes (e.g., a metadata catalog, 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.

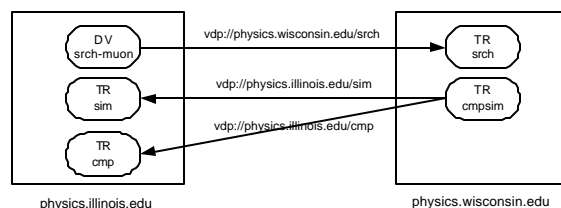


Figure 2: Virtual data hyperlinks between servers (TR = transformation, DV = derivation)

To give one example, Figure 2 depicts a scenario in which transformation and derivation records are distributed across two sites. (The location of the corresponding dataset records is not indicated in the figure.) The “Wisconsin” group defines a compound transformation “cmpsim” composed of two transformations defined by a second “Illinois” group. The first stage of the transformation, “sim”, performs a transformation, while the second stage “cmp” compresses the result. In turn, “Illinois” defines a derivation “srch-muon” that specifies the parameters needed to invoke the Wisconsin particle-searching application “srch” for the particular particle class “muon.”

As this example shows, one consequence of distribution is a need for inter-catalog references, an issue that we discuss below. Furthermore, such hyperlinks need

not be limited to references to remote type and transformation references. Derivation provenance chains can also span across servers, as illustrated in Figure 3, with for example group derivations depend on derivations in a collaboration-wide catalog, and personal derivations depending on those of colleagues.

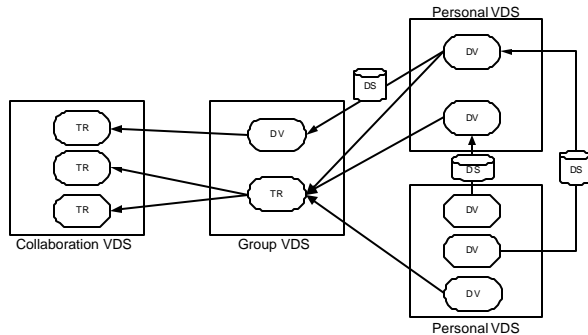


Figure 3: Dataset dependency hyperlinks between objects across virtual data servers

Another consequence of distribution is a need for integration or federation of VDC information contained in multiple catalogs. Given the wide variety of information sources, information qualities, and application information demands that may be encountered in a virtual data grid, we can expect a variety of integration/federation approaches to be useful, ranging from central databases to distributed databases, federated databases, Google-like systems, and peer-to-peer structures.

Some possibilities are illustrated in Figure 4. Four catalogs (“VDSs”) 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.

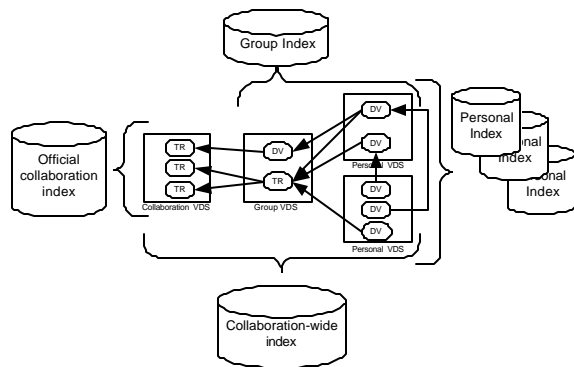


Figure 4: Indexing the virtual data grid at multiple levels

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.

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 virtual data grid 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 virtual data grid 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 in the preceding subsections 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 the Open Grid Services Architecture (OGSA) and implemented by the Globus Toolkit v3. OGSA and its Web services (WS) foundation together address (or will soon 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 virtual data grid. We can now address the central thesis of our vision: 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 virtual data grid 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.

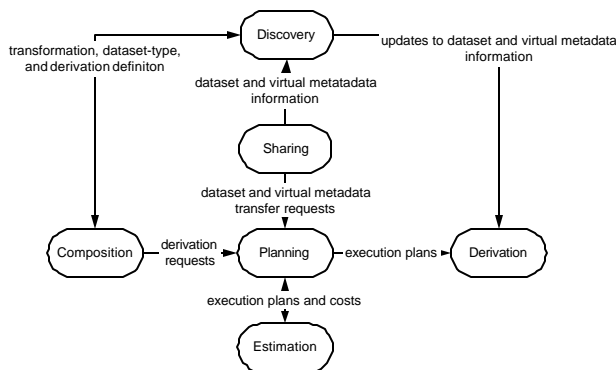


Figure 5: Virtual data process flow

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

- The collaboration has carefully crafted its processing paradigm and created/defined a set of data 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 as compounds of existing transformations.
- Mechanisms to automatically track transformations are integrated into interactive systems.
- Output datasets produced by the execution of derivations are automatically marked.

From an interactive environment a user could trigger the invocation of a derivation, and that this mechanism would run with low overhead and with response time that is as rapid as the speed of the transformation itself (i.e., that batch scheduling systems will ultimately introduce little overhead in the actual process).

5.1 Composition

By composition, we 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 that VDL also being integrated into interactive analysis tools and environments, so that researchers exploring data spaces in an less structured fashion will have the benefits of a historical log of their recent data derivation activities. These users could then choose to snapshot these logs (which could be maintained directly in a virtual data catalog) 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 virtual data catalog, 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 process of generating a database query plan.

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 [18, 19].

The application of procedures to datasets can be performed in a variety of ways, with the following being common patterns.

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 even 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

Estimation is closely coupled with request planning. Given a set of alternative potential plans being evaluated by the request planning function, the estimator must determine the cost of executing the data derivation workflow graph of each plan (which consists of both computation and data transfer nodes). 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 is feasible—whether it can be computed in the time that the user is willing to wait for it.

5.4 Derivation

Derivation refers to the actual execution process of running transformations, with the arguments specified in derivations, to produce specific dataset products. Derivation is conducted by workflow execution management systems that dispatch computation or data transfer requests to specific grid sites, and monitor their completion, dispatching nodes of the workflow graph when the nodes predecessor dependencies have completed. An example of such a schedule is the Condor DAGman facility [15].

The process of derivation produces the invocation records in the virtual data schema which record the precise details of each execution of a derivation – date/time, execution site, and details of the execution environment (OS, processor type, host name, etc).

The automated planning and derivation on a Grid of potentially large and complex workflows can be an important productivity multiplier, permitting VDG users to explore aspects of their data space that were previously inaccessible due to the large burden of planning and managing such jobs. 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

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 and that users may wish to search for data that may exist as “data” and/or in terms of recipes for generating that data.

6 Experience

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, 6]. 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].

The first version of Chimera, Chimera-0, was aimed at obtaining a better understanding of 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 behavior. We also created “canonical” applications that could mimic arbitrary argument passing conventions and file I/O behavior, and used these to create large application dependency graphs to validate our provenance tracking mechanism. The results of these experiments are reported elsewhere [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.

In current work we are implementing challenge problems involving much more interactive analysis processing models than these large, batch-oriented challenges. For both the ATLAS and CMS experiments, we are creating prototype environments where 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, and then produce a series of histograms from the final analysis, and combine these into graphs that can 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 being able to document provenance is well known [20]. Our work builds on preliminary explorations within GriPhyN [2, 6]. There are also relationships to work in database systems [4, 5, 21] and versioning [Marian, 2001 #1819]. 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 is not necessarily 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 virtual data Grids. To date, work in this area has focused on lower-level scheduling issues relating for example to data movement [18, 19].

8 Future Directions

Looking out into the future beyond the ideas and designs proposed in this paper, we can already envision extensions into both significant new capabilities to go further down the path of elevating the level at which users interact with computing resources, and practical extensions to make the design proposed here more realistic and usable.

The most significant new area that we hope to explore is that of extending our virtual information base into a

knowledge base. Using knowledge representation, search, and inference techniques, one could envision raising the level of interaction with the virtual data grid to a very domain-cognizant model, where initial searches, and eventually 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.

Other ideas and avenues of exploration, while less dramatic, are necessary in order to make the design proposed here a robust and usable possibility. These include the following:

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 increasing made to have large collaborations keep all of their data assets, online, in a database.

A model for representing equivalence and similarity between data products needs to be created. 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 behavior and semantics for a certain class of transformations. This will need to be explored and turned into a precise model in order for the VDG to be useful to a broad set of communities, especially in the areas of simulation.

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” – where multiple datasets refer to different overlaid subsets of the same physical storage elements. This raises difficult issues of storages 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 the provenance tracking mechanisms into existing tools, both general (such as SAS, SPSS, Excel, etc) and special purpose, such as analysis environments like ROOT and PAW for high energy physics, or others in other scientific areas.
- Dealing with “update” as an operation a proc can perform on a DS; this maintains provenance but loses re-creatability 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

We will further discuss the limitations of this model and various means to overcome these limitations in section

How does our system apply to “keoliad” systems? (Stonebraker, Gray)

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 the application of this system to several 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 significant 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 Peter Buneman, Raj Bose, Rick Cavanaugh, Peter Couvares, Ewa Deelman, Catalin Dumitrescu, Greg Graham, Carl Kesselman, Miron Livny, and Alain Roy. 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

This section reviews 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 = "${output:a2}";
  exec = "/usr/bin/app3";
  env.MAXMEM = "${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 dl->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};
  exec = "/usr/bin/app1";
}
TR trans2( output a2, input a1 )
{
  argument stdin = ${input:a1};
  argument stdout = ${output:a2};
  exec = "/usr/bin/app2";
}
DV usetrans1->trans1( a2=@{output:"file2"},
a1=@{input:"file1"} );
DV usetrans2->trans2( a2=@{output:"file3"},
a1=@{input:"file2"} );
```

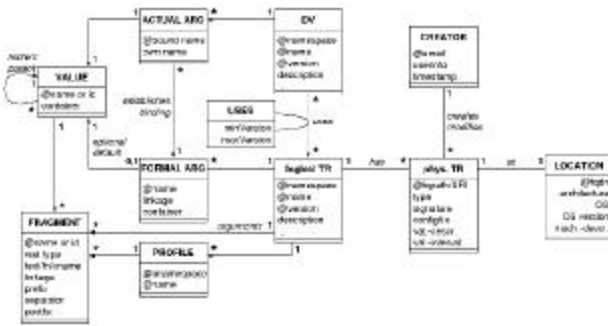
Three simple transformations, and the fourth transformation, trans4, which is a compound transformation composed of calls to trans1,2, and 3:

```
TR trans1( output a2, input a1 )
{
  argument = "...";
  argument stdin = ${input:a1};
  argument stdout = ${output:a2};
  profile hints.pfnHint = "/usr/bin/app1";
}
TR trans2( output a2, input a1 )
{
  argument = "...";
  argument stdin = ${input:a1};
  argument stdout = ${output:a2};
  exec = "/usr/bin/app2";
}
TR trans3( input a2, input a1, output a3 )
{
  argument parg = "-p foo";
  argument farg = "-f "${input:a1}";
  argument xarg = "-x -y -o "${output:a3}";
  argument stdin = ${input:a2};
  exec = "/usr/bin/app3";
}
TR trans4( input a2, input a1, inout
a5=@{inout:"anywhere":""}, inout
a4=@{inout:"somewhere":""}, output a3 )
{
  trans1( a2=${output:a4}, a1=${a1} );
  trans2( a2=${output:a5}, a1=${a2} );
  trans3( a2=${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 )
{
  trans1( a2=${output:a4}, a1=${a1} );
  trans4( a2=${input:a4}, a1=${a2},
a3=${a3} );
}
```

Appendix B - Chimera Virtual Data Catalog Schema – VDL 1.0



Appendix C: Example dataset-type Hierarchy

Dimension: Dataset-format

- Fileset
 - Simple
 - Multi-file list
 - Tar-archive
 - Zip-archive
- Spreadsheet
 - Excel-95
 - Excel-2000
- Relation
 - SQL-table
 - SQL-table-set
 - SQL-table-keyrange

Dimension: Dataset-encoding

- Text
 - ASCII
 - DOS-text
 - UNIX-text
 - EBCDIC
 - MVS Text
 - Unicode
- Table
 - Tab-separated-table
 - Comma-separated-table
- HDF-file
 - HDF-4 file
 - HDF-5 file
- SPSS
 - SPSS-portable
 - SPSS-native
- SAS
 - SAS-transport
 - SAS-native

Dimension: Dataset-content

- UChicago
 - UChicago-student-record
 - UChicago-class-record
- CMS
 - Simulation
 - Zebra-file
 - Geant-4-file
 - Analysis
 - ROOT-IO-file
 - PAW-ntuple-file
- SDSS
 - FITS-file
 - Object-map
 - Spectrometry-raw
 - Image-raw

REFERENCES

1. Annis, J., Zhao, Y., Voekler, 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.
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. Collaboration, T.G. The GriPhyN Project, www.griphyn.org, 2000.
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. OGSA: Grid Services for Distributed System Integration. *Computer*, 35 (6).
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., Voekler, 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. Livny, M. DAGMAN reference.
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. Ranganathan, K. and Foster, I., Design and Evaluation of Dynamic Replication Strategies for a High Performance Data Grid. in *International Conference on Computing in High Energy and Nuclear Physics*, (2001).
19. Ranganathan, K. and Foster, I., Identifying Dynamic Replication Strategies for a High Performance Data Grid. in *International Workshop on Grid Computing*, (2001).
20. Williams, R., Bunn, J., Moore, R. and Pool, J. Interfaces to Scientific Data Archives, Center for Advanced Computing Research, California Institute of Technology, 1998.
21. Woodruff, A. and Stonebraker, M. Supporting Fine-Grained Data Lineage in a Database Visualization Environment, Computer Science Division, University of California Berkeley, 1997.