Datagridflows: Managing Long-Run Processes on Datagrids

Arun Jagatheesan[1,2], Jonathan Weinberg[1], Reena Mathew[1], Allen Ding[1], Erik Vandekieft[1], Daniel Moore[1,3], Reagan Moore[1], Lucas Gilbert[1], Mark Tran[1], and Jeffrey Kuramoto[1]

[1] San Diego Supercomputer Center,
University of California, San Diego,
9500 Gilman Drive, MC0505, La Jolla, CA 92093
{arun, jonw, alding, moore, iktome, kuramoto}@sdsc.edu
{rmathew, evandeki}@cs.ucsd.edu
mixx@umail.ucsb.edu
mdtran@ucsd.edu
[2] Institute of High Energy Physics and Astrophysics,
University of Florida,
Gainesville, FL 32611
[3] Department of Computer Science,
University of California, Santa Barbara, CA93106

Abstract. This paper is an introduction to *Datagridflows*. Until recently, datagrids were generally considered over-hyped and the associated technologies not widely embraced in the academic community. Today, datagrids have become a reality and an important technology for managing large, unstructured data and storage resources distributed over autonomous administrative domains. The datagrids that are operating in production provide us an idea of new requirements and challenges that will be faced in future datagrid environments. One such requirement is the coordinated execution of long-run data management processes in datagrids. We term these processes as "datagridflows". This new area provides exciting opportunities and challenges to researchers in distributed computing and distributed databases. This paper is intended to introduce these challenges to other researchers, including those new to grid computing. We provide motivation through discussion of datagridflow requirements and real production scenarios. We introduce current work on datagridflow technologies including the *Datagrid Language (DGL)* for describing datagridflows in datagrids.

1 Introduction

Datagrid technology is currently used for managing very large, unstructured data storage resources [1, 2, 3]. The need for long-run data management processes on top of datagrid environments is seen as a common emerging requirement in most datagrid deployments. Examples of these long-run processes include datagrid information lifecycle management, datagrid triggers, and data-intensive computational workflows. These long-run processes could be considered "datagrid workflows" and are discussed later in this paper. We refer to these long-run datagrid processes as datagridflows.

J.-M. Pierson (Ed.): VLDB DMG 2005, LNCS 3836, pp. 113 – 128, 2005.

In the following section, we introduce some fundamental concepts in datagrids for the benefit of those new to grid computing. In section 2, we describe three motivating scenarios for datagridflows and our work on the Data Grid Language. We discuss the requirements and components of a system to manage datagridflows in section 3. In section 4, we provide some overview of our work on the Data Grid Language as part of the SRB Matrix Project. Related and future works to this paper are presented in section 5.

1.1 Data Grid Landscape

In this section, we introduce datagrids, associated concepts and relevant terminology to prepare the reader for the problem statement discussed in the following sections.

Grid Computing. We describe a "grid" as a coordinated distributed computing infrastructure, formed by combining heterogeneous resources from autonomous administrative domains. Grids provide the infrastructure that is used for large-scale, resource-intensive, and distributed applications. The definition of a Grid is continually evolving as different people have different perspectives of the same technology. The commonality that is observed in the different perspectives of the "Grid" is the formation of a logical infrastructure as a single ensemble, by dynamically combining independently managed resources.

Datagrid. A datagrid is a logical unified view of a grid's data storage infrastructure. Data storage middleware create a federated, location independent, logical infrastructure namespace that dynamically spreads across the grid's administrative domains. Datagrids support sharing data collections and storage resources between autonomous administrative domains. A shared collection is a logical aggregation of digital entities, (e.g.) files, which are physically distributed in multiple physical storage resources that are owned by multiple administrative domains. A shared resource allows users from multiple administrative domains to share data storage space. The core concept behind the success for datagrid software is the concept of "data virtualization".

Data Virtualization. Data Virtualization is the concept of bringing together different heterogeneous data and storage resources into one or more *logical views* so that the distributed and replicated data appear as a single logical data source managed by a single data management system. This logical view is simple for users and applications as it hides the complexity of working with distributed and heterogeneous systems. The logical view is provided on top of a logical resource namespace, allowing high levels of flexibility for distributed computing and migration of data storage resources. Data and resource names are logical and can be physically changed or migrated without affecting the applications. The underlying concept behind the datagrids and data virtualization is the same as the concept behind relational databases: *to isolate physical organization of the data from logical schema.* In data virtualization, we go one step further. Instead of completely hiding the physical organization of the storage resources where the data resides, another logical namespace of storage resources is provided to the applications. Applications now have the added capability to perform distributed data management operations on the combined logical data namespace

along with logical resource namespace without having to directly interact with the physical storage resources or the physical organization of data.

There has been a significant increase in use of datagrid technology over the past few years. Data storage infrastructures using datagrid technologies are deployed in many countries. Much of the data managed by these technologies is in the form of files. One of the popular datagrid management systems (DGMS) [1], the SDSC Storage Resource Broker (SRB) [2], is believed to broker around a Petabyte of data worldwide at the time of this writing.

Multiple independent organizations deploy the SRB middleware on top of their existing physical storage resources without any changes to the existing system. The existing physical storage resources are represented in the SRB datagrid namespace as logical storage resources. Each SRB storage server that runs on top of a physical storage system maps that particular physical storage system into the data grid logical resource namespace. Many organizations participate in a data grid. Users can view and use the resources of users from other organizations given appropriate access permissions and authentication mechanisms. Users use any logical resource from the data grid logical resource namespace using the SRB protocol without even knowing where the resource is physically located or what type of physical storage resource is actually used. In addition, users can create an aggregated logical view of distributed data in the form of shared collections, enabling them to have the same logical namespace or data organization even if when the data is moved. Thus, the data namespace or the logical view of the data in the grid is independent of infrastructure and location information for the end users.

2 Long-Run Processes in Datagrids

The widespread use of datagrids has helped us observe several common usage patterns in datagrid environments that require long-run datagrid processes. In this section, we present the three prominent patterns that we have observed. These motivate our work on datagridflows.

2.1 Data Grid ILM

Information Lifecycle Management (ILM), as described in the data storage industry, refers to the dynamic re-orientation of data placement and data retention strategies based on storage cost and the "business value" of the data to be managed. The term "business value of data" refers to the value certain data or information provides to the business requirements. Unlike traditional Hierarchical Storage Management (HSM) solutions, which normally use "data freshness" as the most important attribute in determining data placement, ILM solutions use data value and business policies to determine data placement and retention. It must be mentioned that in most business cases a high value of data freshness will automatically yield a high business value for the data. Hence, ILM could be considered an extension of HSM.

In a datagrid, information in the form of several related data collections would have a lifecycle that spans multiple organizations. Information in the datagrid could

be created by one organization, accessed or replicated by other organizations, and archived at yet another organization before finally being deleted from the datagrid.

During its lifecycle, information in the grid would have different business values for different domains participating in the datagrid. This value is based on the needs of a particular domain's users and the role played by that domain in the data grid. For example, data being created might be of interest to the domain that is creating it. Later, some other domain in the data grid might have more value for the same information. We refer this as domain-specific value as "domain value". Organizations could create replicas of the same data in their own domains as the domain value of certain data grows. Once a domain's users are not interested in some information, its domain value decreases and data can either be deleted or migrated to less expensive storage systems. A change to data storage organization with respect to domain value of some information is called a "datagrid ILM processes". These changes usually do not involve any transformation of data. They could involve replication, migration or removal of existing data, changing access permissions on some data before they are migrated or archived, etc.,

In addition to changes in the domain users' interest in information that could initiate the ILM processes, the role played by a domain in the datagrid could also initiate ILM processes. In some cases, one of the domain's roles in the data grid could be just to archive all or some selected information in the datagrid. This could be a third-party service provider or an IT department for the enterprise responsible for archiving data. The archiver domain might not have any real users who are interested in the information – but its business processes are interested in archiving the information. The archiver domain could store the information for years, before finally moving it the lowest cost data storage system from a long-term storage management perspective. The archiver domain could be an example for what we refer to as an "imploding star". Information from all the domains in the datagrid is finally pulled towards this domain. This certainly involves a very well planned archival schedule. An example for this type of imploding star is the BBSRC-CCLRC data grid [3]. In the BBSRC project, information from multiple hospitals in United Kingdom are finally archived into an archiver site.

The complement of the imploding star based datagrid ILM is the "exploding star". In this case, information is pushed or replicated outside the domain of its creation. For example, the datagrid created for the CMS High Energy Physics experiment at CERN has many domains that require the data generated by the CMS experiment to be replicated in stages at different tiers across the globe. The CERN domain thus acts as the exploding star. Domains can play other roles such as a "data curator" role in a digital library that is powered using the data grid technology.

We can observe some commonalities and generic requirements in these datagrid ILMs. All of them require long-run processes on top of the datagrid namespaces. These long-run processes could be started, stopped and restarted at any time. For example, an ILM process could only be run at some domains during non-working hours or on weekends. This would require powerful and highly flexible systems to manage these datagrid ILM processes. A requirement from digital libraries and persistent archives, like the National Archives Persistent Archives Test bed (NARA PAT) [4] is to preserve the provenance information associated with these ILM processes. The requirement is to enable the storing of provenance information for not only the

DGMS operations performed by the system, but also the operations that are performed as part of the archival pipeline.

Currently, some simple datagrid ILM processes can be implemented using simple scripts and cron jobs on some operating systems. System administrators are familiar with these scripts. However, once the requirements include multiple domains, multiple system administrators and multiple ILM processes, more sophisticated systems are required to handle problem. The proposed new systems for datagrid ILM must support:

- Start, stop, pause and restart of datagrid ILM Processes
- Query the status of the any datagrid ILM any time
- Provenance information of all the processes managed at any time even (years) after the execution
- Programmatic API to define these datagrid ILM and programmatic interface for interaction by other systems
- Programmatic API to query and monitor any step in the datagrid ILM process

One major requirement is to provide an interoperable description of the datagrid ILM processes. A standard format could be used across all the related systems like datagrids, grid file systems, digital libraries, persistent archives and dataflow systems. Such a standard based on an XML Schema would allow programmatic interaction of all the systems. The proposed XML schema must support the definition of ILM processes of various complexities. The schema must describe all relevant entities, including data, resources, and users. The schema would have to be programmatically described and executed dynamically as the constraints associated with these processes are dynamically modified.

2.2 Datagrid Triggers

The datagrid namespace is a logical view of data and storage resources. A datagrid trigger is a mapping from any event in the logical data storage namespace to a process initiated in the datagrid in response to such an event. Datagrid triggers are defined on top of the datagrid namespace and could have the following components.

Event. An event could be any change in the datagrid namespace including updates, inserts, and deletes. Datagrid triggers could be triggered before or after events complete. Unlike database transactions datagrid processes or not transactional. The results of applying the trigger-based mechanisms on this non-transactional, large-scale, distributed data management system have not yet been studied.

Condition. Trigger execution is determined by the evaluation of some state information in the datagrid. This is very similar to the database Event-Condition-Action (ECA rules) based processing used in database rules.

Actions. An Action is the execution any data management process on the datagrid namespace. Multiple actions could be performed based on the evaluation of the condition associated with the trigger.

Datagrid triggers will play an important role for managing unstructured data in datagrids. Simple use-cases include: creating metadata when a file is created, sending notifications when specific types of files are ingested, and automating replication of certain data based on their meta-data.

Datagrids allow user-defined metadata to be associated with data. Triggers could make use of these parameters. There are many open research issues here. Datagrid management systems (DGMS's) [1] will allow multiple users to define triggers. Different results might be produced based on the order in which triggers defined by multiple users are processed for the same event. Further complicating the situation is the non-transactional nature of datagrid processes.

In databases, the Structured Query Language (SQL or PL/SQL) can describe the triggers and the DBMS executes associated actions. A similar language is required for DGMS's to describe triggers with respect to files, the metadata that are associated with those files, data collections, data storage resources, etc. Such a language should support data types such as collections and datasets. The proposed language could also be used to describe constructs in datagrids similar to "stored procedures" in databases. This will allow the datagrid stored procedures to be run from the DGMS itself rather than executing the procedure outside the DGMS using client side components. We introduce "Data Grid Language" (DGL) as a possible solution for this later in this paper.

2.3 Data-Intensive Workflows

The last motivation that we want to mention regarding long-run processes in datagrids is the use of the datagrid infrastructure to perform scientific or computational workflows on unstructured data. Such workflows are sometimes referred to as "scientific workflows" because they are often used in certain scientific applications, but the associated concepts apply equally for non-scientific workflows that require intensive processing.

Grid-workflow is the automation of a business process whereby data and tasks are passed from one grid-participant to another according to some set of procedural rules. A single grid workflow process could have multiple tasks that might have to be executed at different domains participating in the grid. The dynamic scheduling of these tasks to the different participating domains could be based on the combined cost all the tasks together at different domains. The cost of executing each task at a domain could be based on multiple parameters including the amount of data moved, the number of CPU cycles that would be left idle in the grid, the clock time taken to execute all the tasks, the bandwidth utilized, etc. The cost is just an approximate value based on certain heuristics used by the scheduler.

During their execution, Grid-workflows must consider different logics: the business process logic, the execution logic and the infrastructure logic as explained below.

Business Logic. Business logic is a representation of the specific business task that takes part in the workflow. Some examples of business logic are: processing an order-entry form (e-business), determining a document type while archiving it in the prototype for National Archives Workflow (document management), or any transformation used in scientific pipelines (scientific workflow). The isolation of the business logic

from the complexities involved in datagrid computing provides ease of development of the business logic. The business logic development team need not be concerned with scaling up its solution or taking advantage of the distributed nature of the datagrid. They should only be required to describe the requirements in terms of resource types and the service levels required to execute the business logic. Business logic is usually in the form of binary executables that could be run on appropriate platforms in the datagrid.

Infrastructure Logic. Infrastructure Logic refers to the logic that has to be used while matching the tasks in the workflow with the appropriate resources and domains within the grid infrastructure. Infrastructure logic would involve the description of available resources in the infrastructure, the service level agreements (SLAs) the resources can support, the preferred type of users or tasks that could be executed on each resources, etc. Infrastructure logic could also involve heuristics that are supposed to be used by a Datagridflow Management System (DfMS) while scheduling the tasks to the different resources in different domains.

The DfMS would have to map the requirement of each business logic task to the appropriate resources required. The workflow description would dictate what types of resource are required at what SLAs. The description might be just a logical or abstract specification of the type of resource required rather than a specific physical system. This allows dynamic binding to a particular resource at runtime. The workflow description is used by DfMS along with Grid Resource Brokers to bind the task with appropriate resources. For example, the workflow description might logically specify that a particular task would require an archival system, a high-performance file system, or a certain number of compute nodes. Infrastructure logic on the other hand, would specify the mapping from these logical resource types with the physical endpoints and the SLAs that can be supported. The system administrators could change the infrastructure logic based on their own domain requirements, assuring them full autonomous control over what resources are shared with other grid users and at what SLAs.

Execution Logic. Execution logic provides the control-flow and ordering of tasks that take part in the workflow. Execution Logic is provided by the end-user or the workflow designer. It provides a description of the workflow execution, identifying the tasks that take part in the workflow, the order in which they should be executed, the relationship among them, their input and output data sets, etc.

Execution logic also has information on the state of execution. This information can be checked before execution of any process. Fault handling information for the processes could also be provided in the execution logic. Execution logic could remain independent of the infrastructure dependencies allowing late binding of resources. However, a workflow designer could still choose to specify a particular resource instead of leaving it abstract to be bound later.

Execution logic also captures the requirement to run tasks for a specified number of times or until some milestone is reached. This is very useful in datagrids where the workflow involves iterating some set of tasks over collections of files. The files are used as input data and processed according to a datagrid query, which could be part of

the execution logic itself. This allows configuration of runtime parameters by changing the execution logic rather than configuring the business logic and recompiling the associated code. The execution logic could be viewed as the abstract definition of a workflow without concrete descriptions of the underlying physical infrastructure.

Infrastructure-based Execution Logic. The Execution Logic is converted dynamically into Infrastructure-based Execution Logic just before the execution of the tasks that are described in the workflow. This is a multi-stage hierarchical process. An analogy for this process could be the query re-writing or optimization of SQL before a final query plan is generated and executed by the databases. The description of the execution logic is rewritten into infrastructure-specific execution logic based on multiple factors including: the requirements of the task, availability of resources, the physical locations of the input or output data, the presence of "virtual data" [5] or "virtual services" [6] and other infrastructure heuristics.

Iterations or milestones present in the execution logic would require a small section in the description of the execution logic, a group of tasks, to be dynamically converted into infrastructure-based execution logic multiple times. The group of tasks, a small section of the execution logic for a single iteration, would have to be dynamically converted into infrastructure-based execution logic very late in the processes just before execution. This late binding allows execution of the each iteration at a different location based on the infrastructure availability just before the tasks are executed.

The scheduling or selection of the appropriate resources for each task has to choose the location for execution of a task based on: the available physical locations of input data (replicas), desired physical location of the output data, location of the business logic (code) and the available resources where the task can be executed. If the required output data is already available (virtual data), it need not be derived again. The final infrastructure-based execution logic for each task would have the chosen replica to use as input, the location of the output data and the grid resource to use. In a datagrid, the replica selection could be handled by the DGMS itself based on location of execution of the process.

All the execution logic associated with the Grid-workflow must be generated programmatically and exchanged among the participating resources. This includes the datagrid execution logic and infrastructure-based execution logic. The Data Grid Language described in the following sections could be used for to describe these sets of logic. Even though multiple workflow languages are already available, the existence of datagrid-related data types and operations as part of the language itself makes it the suitable language to describe these grid-based data-intensive processes that take part in scientific workflows.

In this section, we have surveyed three of our major motivating scenarios in detail and their requirements with respect to datagrid technology. One common observation from all these scenarios is the need for datagridflows on top of datagrid systems. Another requirement that has been mentioned in all the scenarios is the need for a language to describe the long-run processes in the datagrid. In the next section, we introduce Datagridflows and their requirements.

3 Datagridflows

Datagridflow is the automation of a long run process whereby data and/or tasks are passed from one datagrid participant to another according to a set of procedural rules. Datagridflows are data-intensive long run processes like datagrid ILM, datagrid triggers, or computational workflows in a datagrid environment. Datagridflows could be viewed as a subset of regular workflows that involve long-run processes on datagrids. Most of the data processed is unstructured, file-like data.

Workflow systems have been around for many years. There are many ways to hard-wire workflows and develop a system that uniquely satisfies a single user's requirement. This approach is easy for the developers to begin with as they can use any of their favorite programming languages to hard-wire the tasks involved in the workflow. However, from a long-term perspective, this approach is not optimal and it becomes extremely expensive to maintain the code that supports the whole system. Any change in the execution logic or the infrastructure logic would require modification of the whole system. A generic system would be useful for the datagrid community, which has clear needs to manage datagridflows, as can be seen in multiple projects including National Archives Persistent Archive Test bed Project [4], Southern California Earthquake Center [7], CCLRC-BBSRC project [3] and LLNL UCSD SciData Management Pipeline.

3.1 Generic Requirements for a Datagridflow Management System (DfMS)

The challenge is to provide a generic system that can manage most of the datagridflow requirements faced by these data-intensive projects. The common patterns that we observe from our users' requirements when they want to manage their datagridflows:

- *Data-intensive flows:* Most of the projects that use datagrid technology usually have large data collections. DfMS must take full advantage of the underlying DGMS software that provides all the functions required to manage the very large unstructured data.
- *Scalability:* DfMS must be scalable in terms of the number of tasks within a single workflow; number of workflows that can be processed, and the number of resources the workflows can physically take advantage of to complete a workflow.
- *Collections and Files:* Most of the data that is processed in a DfMS is in the form of collections and files. DfMS's must support these data types and the operations that can be supported on collections and files in a datagrid.
- *Highly Flexible:* Most of these projects will deploy the DfMS in production for at least five years. Over this time, many requirements, probably unknown during requirements analysis, will emerge. The system should therefore be flexible to handle new requirements.
- *Cost of Operation:* Having one more software system to manage increases the Total Cost of Operation (TCO) of the project. DfMS must minimize the maintenance requirements and the system administrator should not have a need to learn another system.

- *Provenance:* DfMS must have manage information about all workflows and their tasks. This information would be queried and audited later.
- *Novice and Expert Users:* DfMS must have a GUI-based system to interact with novice users and an API based interface for developers and expert users to programmatically interact with the DfMS
- *Distributed Grid Infrastructure:* DfMS must take advantage of the distributed grid infrastructure while executing its operations
- *Task Granularity:* Workflow designers should have the flexibility to design datagridflows with each task that is not too small and not too large to be called a task.

The above requirements are generic for both business and academic/scientific workflows. Similar business use cases would be observed once business users start using datagrids and the Grid File System (GFS) [9].

3.2 Components of a Datagridflow System

The following are the components of a hypothetical Datagridflow System from a high-level perspective.

Datagridflow IDE (GUI). A Datagridflow modeling interface would serve as an Integrated Development Environment (IDE) for end-users to interact with the DfMS. A modeling markup language describes datagridflows and stores it locally for the users to use again or view the datagridflow rendered on the IDE. MoML [8], used in Ptolemy II/Kepler uses, this approach to serve as a datagridflow IDE.

Datagrid Language (DGL). A language to describe, query, and manage execution logic and infrastructure-based execution logic. The SRB Matrix uses this approach. A DGL document could be created by the IDE and sent to the DfMS server for processing. More on DGL is provided in the next section.

DfMS Server. The DfMS server can service DGL requests both synchronously and asynchronously. DfMS server manages state information about all the tasks, which can be queried at any time. The DfMS server works on top of the datagrid server (DGMS) and can support the datagrid operations provided by DGMS. In the SRB Matrix project [10], the *Matrix* Server uses SRB as its DGMS. Multiple DfMS servers can form a peer-to-peer datagridflow network with one or more lookup servers. DfMS servers could have additional capabilities to directly interact with the DGMS server, allowing the users to create Datagrid Triggers and Datagrid ILM jobs at the DGMS it self. The DfMS server can provide the concepts of virtual data by incorporating a virtual data system as a component. The GriPhyN Chimera System is an example of such a component that could be present in the DfMS server.

Infrastructure Description Language. The Infrastructure Description Language describes the infrastructure at each domain and the different SLAs they can support. Infrastructure includes data storage resources, compute resources, DGMS server location etc.

Grid Schedulers and Brokers. Grid schedulers and brokers act as intermediaries, that do the planning and matchmaking between the appropriate tasks in a workflow with the resources that are available. They are used to convert the abstract execution logic into concrete infrastructure-based execution logic. Tools are available for planning and scheduling on the grid. One such tool is the GriPhyN Pegasus planner [11].

4 The Data Grid Language

We have discussed the need for a datagrid language as part of our motivating scenarios. Just as SQL is used for databases, an analog is needed for datagrids. Our contribution to the datagridflows and the datagrid community is the *Datagrid Language* (DGL), which is useful for all of our motivating scenarios. DGL is an XML-Schema specification that can be extended for domain-specific operations and used by any community.

DGL explicitly supports data types such as datagrid collections, files and datagrid operations as part of the language it self. This enables the description of file-based flows and datagrid collection processing. DGL can be used to describe datagridflow processes, queries, and status. The language is designed to work with a protocol based on a request-response model. In addition to request-response, DGL can also be used with one-way messages also. The requests can be synchronous or asynchronous.

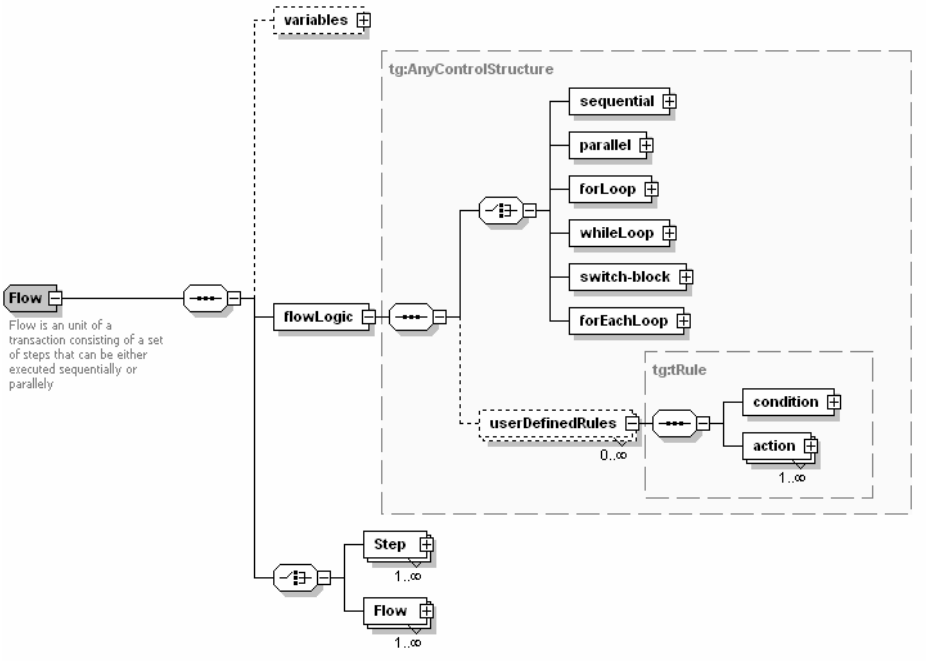

Fig. 1. Structure of a Flow

DGL describes each task of a datagridflow as a "Step" with associated input and output parameters. One or more steps are aggregated into "Flows", which are recursive control structures that describe how to execute steps. Each flow is like a block of code in modern programming languages with its own variable scope, commands, and steps. Each flow defines a unique control pattern that dictates how its contents should be executed, e.g. sequentially, in parallel, while loop, for-each loop, switch-case, etc. These patterns are very similar to any modern programming language. Using these control structures recursively, users can create arbitrarily complicated gridflow descriptions. Figure 1 shows the schema definition for a flow in DGL.

Each DGL transaction generates a unique identifier that can be used to query the status of the any task in the workflow at any level of granularity. The identifier for any particular task or flow can be shared with all other processes that require access to the status of the particular task or flow.

DGL also supports user-defined, Event-Condition-Action rules. This enables an event-based model for datagridflow programming. More information about DGL can be found in Appendix A of this document.

DGL has been used in prototype runs for managing datagridflows at the UCSD Libraries and SCEC Project. Datagridflow for data-integrity and MD5 calculation was described in DGL and executed by SRB Matrix servers for the UCSD Library data. SCEC workflow for ingesting files into the SRB datagrid was also performed using DGL [14].

5 Related and Future Work

Multiple efforts are underway to tap the power of the grid infrastructure and to manage long run process or workflows. There are clear differences in the objective and/or approach taken by each of these efforts. Some of the projects working in related areas are mentioned here.

GridAnt [12] is a client-side workflow engine that provides scripting support to initiate and manage the workflow. The state information of the workflow is managed at the client side. GriPhyN Pegasus [11] could be used as planner in a grid workflow to avoid redundant computation of existing data products. Pegasus is used as a component in GriPhyN Virtual Data System [5]. Kepler [13] is an effort to provide an extensible IDE and full system for scientific workflow (which are also long run processes). Additionally, there are multiple workflow related efforts, which are based either on Web/Grid Service Composition or on Process Ordering.

Our current work in the SRB Matrix Project is to support our existing SRB users with these datagridflow requirements. We are also working on providing a rich GUI (IDE) to DGL using VERGIL GUI (used in Ptolemy II and Kepler). The user interface will be defined by the MoML modeling language, with execution taking place using the DGL.

There are many research issues that would be interest to others, including:

- Peer-to-peer datagridflow network and its protocols
- Distributed data scheduling for datagrid ILM policy strategies for enterprises
- Dynamic datagrid scheduling based on heuristics at different domains

Datagridflows is an emerging field that presents some exciting challenges. Datagrid users already require powerful peer-to-peer datagridflow networks. More work would help the community understand more about the requirement and the usefulness of different approaches taken.

6 Conclusions

Datagridflow is an emerging field that supports the proliferation of datagrid technology by addressing the new requirements of datagrid users. Datagridflows enable users to automate or semi-automate tasks in the datagrid. Many more challenges and opportunities are present for researchers from distributed computing and distributed databases.

Acknowledgement

This work was supported by NSF GriPhyN, NPACI REU and SDSC REU. We would like to acknowledge Peter Berrisford of CCLRC, UK; David Little, UCSD Libraries; and Marcio Faerman and Phil Macheling of the SCEC project for providing us descriptions of their datagridflows requirements in their projects.

References

1. Moore, R.W., Jagatheesan, A., Rajasekar, A., Wan, M. and Schroeder, W., "Data Grid Management Systems," *Proceedings of the 21st IEEE/NASA Conference on Mass Storage Systems and Technologies*, 2004, Maryland.
2. Rajasekar, A., Wan, M., Moore, R.W., Jagatheesan, A. and Kremenek, G., "Real Experiences with Data Grids – Case-studies in using the SRB," Proceedings of 6th International Conference/Exhibition on High Performance Computing Conference in Asia Pacific Region (HPC-Asia), December 2002, Bangalore, India
3. BBSRC-CCLRC Data Grid. Web site: (http://www.e-science.clrc.ac.uk/web/projects/bbsrc_grid_support)
4. Archivist Grid Website: http://www.sdsc.edu/Press/2004/04/040904_PersistenArchives.html
5. Foster, I., Voeckler, J., Wilde, M. and Zhao, Y., "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation". In Scientific and Statistical Database Management, (2002).
6. Jagatheesan, A., Moore, R., Rajasekar, A. and Zhu, B., "Virtual Services in Data Grids", In the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC), July 2002, Scotland.
7. Southern California Earthquake Center, SCEC: http://www.scec.org/cme
8. Edward A. Lee and Steve Neuendorffer. *MoML — A Modeling Markup Language in XML — Version 0.4*. Technical report, University of California at Berkeley, March, 2000
9. Arun Jagatheesan, "Architecture of Grid File System", Gridforge. https://forge.gridforum.org/projects/gfs-wg
10. SRB Matrix Website: http://www.sdsc.edu/srb/matrix

11. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi and M. Livny , "Pegasus: Mapping scientific workflows onto the grid," Across Grids Conference 2004, Nicosia, Cyprus.
12. Kaizar Amin and Gregor von Laszewski, GridAnt: A Grid Workflow System. Manual, February 2003 http://www-unix.globus.org/cog/projects/gridant/
13. B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, Y. Zhao, "Scientific Workflow Management and the Kepler System", Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows.
14. Weinberg, J., Jagatheesan, A., Ding, A., Fareman, M. and Hu, Y., "Gridflow Description, Query, and Execution at SCEC using the SDSC Matrix, " Proceedings of the 13th IEEE International Symposium on High-Performance Distributed Computing (HPDC), June 4-6, 2004, Honolulu, Hawaii, USA.

Appendix

A Structure of DGL

A DGL document is a XML based description that could be either a *Data Grid Request* or *Data Grid Response*. A Data Grid Request is sent from a client to the DfMS server. Currently, the DfMS server uses a request-response paradigm and replies with a Data Grid Response for each request.

Figure 2 shows the structure of a Data Grid Request. It contains general information including: Document metadata, Grid user information and the Virtual Organization to which the user belongs. The Data Grid Request's core component is either a *Flow* or a *FlowStatusQuery*. A *Flow* describes a workflow to be executed and a *FlowStatusQuery* is a query on the status of execution of a *Flow* at any granular level.

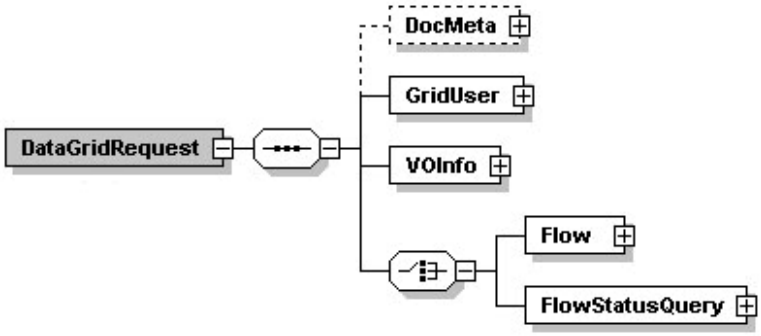

Fig. 2. Structure of a DataGridRequest

The *Flow* is a recursive data structure that represents the gridflow execution. It contains more recursive *Flow*s or *Step*s (called its "children"). Abstractly, we can think of a *Flow* as an execution environment or a block of code that sets up a scope and behavior for its children execute (e.g. sequentially, in parallel, for-loop, etc).

As shown in Figure 1, each *Flow* contains three sections:

- *Variables* – A *Flow* can declare any number of variables for use in its scope
- *FlowLogic* – This component dictates the logic by which the contents should be executed (e.g. sequentially, in parallel, etc)
- *Children* – Sub-flows or steps (but not both), which will be executed within this *Flow*'s scope according to its *FlowLogic*.

FlowLogic

The *FlowLogic* element contains two sections: the first is a choice of control structure (e.g. sequential, parallel, etc) that dictates how the children of this *Flow* will be executed. The second is a set of *UserDefined Rules* that encapsulate the actions that the *Flow* should take upon starting up and before exiting. Figure 3 shows the *FlowLogic* schema.

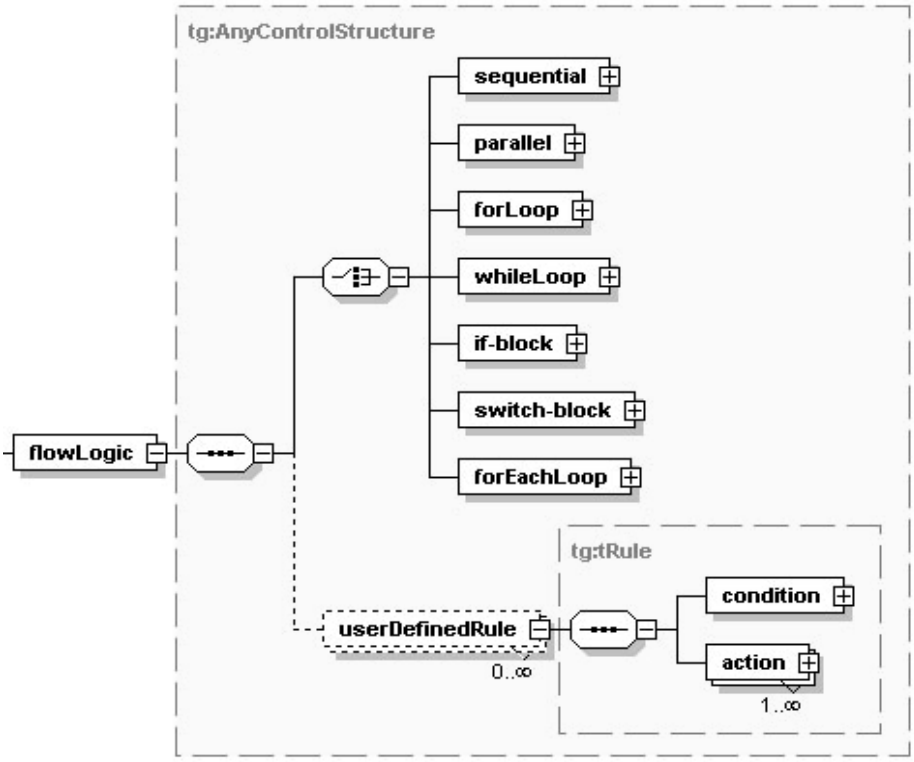

Fig. 3. flowLogic schema

Before it starts execution, a *Flow* will execute the user-defined rule named "beforeEntry" if one is defined in its *FlowLogic*. After finishing execution, it will execute the rule "afterExit" if one is defined.

User Defined Rule

A *UserDefinedRule* is similar to a switch statement in programming languages. A *UserDefinedRule* consists of a condition and one or more action statements. The condition is represented using *tCondition. Tcondition* is a usually simple string that is evaluated. It is possible to use DGL variables in the Tcondition. Each *UserDefinedRule* has one condition and can have one ore more Actions. Each action has a (string) name associated with it. The Actions are executed if the condition statement evaluates to the name of the action.

Step

A S*tep* is a concrete action that a gridflow performs. A S*tep* can declare variables and *userDefinedRules* just like a *Flow*, but contains a single element called an *Operation*. The operation describes some atomic operation that the gridflow is to execute. DGL supports a number of DataGrid related operations for SDSC's Storage Resource Broker (SRB) or execution of business logic (code) by the DfMS server.

Data Grid Response

A Data Grid Response is sent by the DfMS to the client for every Data Grid Request. The design for Data Grid Response facilitates both synchronous and asynchronous requests. Synchronous Data Grid Requests are replied after the execution of the flow with a Data Grid Response that contains the status of flow. Asynchronous Data Grid Requests are replied with a Request Acknowledgement inside the Data Grid Response. Request Acknowledgement contains a unique identifier for each request and the initial status of the request and its validity. Clients can use this identifier to get the status of the execution of the flow. The figure below shows the structure of a DGL Data Grid Response.

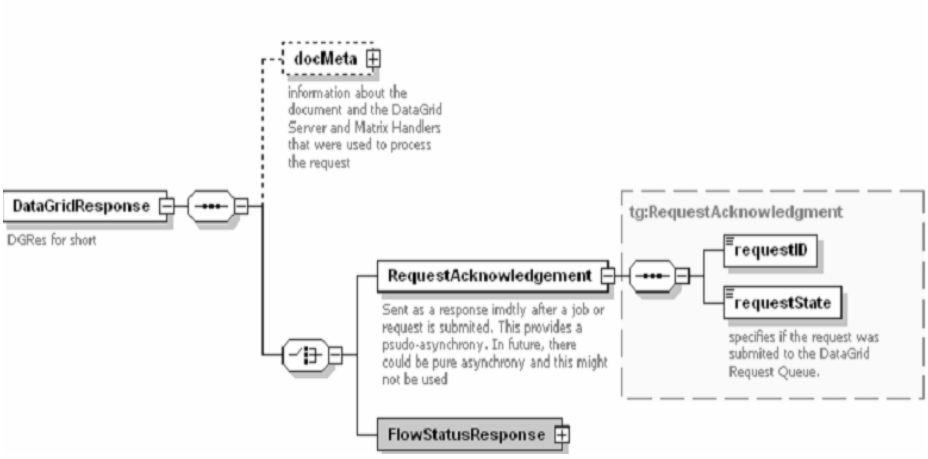

Fig. 4. Data Grid Response