

Active Data: Supporting the Grid Data Life Cycle

Tim Ho and David Abramson

{tim.ho, david.abramson}@infotech.monash.edu.au

Monash e-Science and Grid Engineering Lab

Faculty of Information Technology,

Monash University

900 Dandenong Road, Caulfield East, 3145, Australia

Abstract

Scientific applications often involve computation intensive workflows and may generate large amount of derived data. In this paper we consider a life cycle, which starts when the data is first generated, and tracks its progress through replication, distribution, deletion and possible re-computation. We describe the design and implementation of an infrastructure, called Active Data, which combines existing Grid middleware to support the scientific data lifecycle in a platform-neutral environment.

1. Introduction

The 21st century has been described as the century of data. Various groups predict an exponential growth in the amount of data that will be captured, generated and archived. For example, Foster predicts a ten fold increase every five years from 2000 to 2015 alone [14]. Not surprisingly, there are numerous projects addressing the computer science challenges that underpin management of this explosion in the amount of data.

Typically, data is generated in two main ways; either from real laboratory based experiments or in-silico experiments. Real experiments in areas such as high energy physics, astronomy, biology and medicine, and geophysics, capture data from scientific instruments like telescopes, micro-arrays, seismic sensors and synchrotrons. The data is usually stored in primary repositories, where additional metadata can be attached and made available for interrogation. Primary data can be replicated and distributed to a variety of geographically distributed sites, where they are processed, mined, analyzed and refined to produce secondary data sets. Primary data also comes from in-silico experiments; computer simulations that model some real world phenomenon (possibly driven by real world data) and produce output data which is then stored and analyzed in much the same way as real data. Regardless of the data source, there is a well

recognized need for data curation and provenance information management [17][31]. For example, metadata such as the date and environmental conditions must be saved for a real experiment. Likewise, input parameters and date information must also be saved for an in-silico experiment. Because data from these two sources is handled in very similar ways, data management middleware typically does not distinguish between them. Thus, common solutions for handling metadata can be applied regardless of source. Tool kits such as the Storage Resource Broker (SRB) from the San Diego Supercomputing Centre [1], and the Globus RLS [22] epitomize this approach.

The problem with an exponential growth in data is that it requires an exponential growth in storage capacity. Conventionally, communities usually store and replicate everything they capture (or compute). However, this is clearly wasteful, and it is unlikely that we will be able to continue doing this for very long. There are a number of potential solutions to the storage problem. For example, compression techniques might reduce the amount of storage required. However, in some instances it may be simpler to recapture or re-compute the data, rather than to store it. This leads to the development of a *virtual data grid* [16], where data are identified by logical names and derived data from previous experiments can be re-generated when needed. Research on virtual data has attracted focused attention from a small number of groups (e.g. [9][15][21]).

One view of the re-computation problem is that we want to produce a copy of a file transparently where one does not exist. Thus, if an application opens a file that was produced by a computational model, but the data has been deleted, then the file could be re-computed without the application being aware. Clearly, implementing such a strategy requires sufficient metadata on how a data set is produced, and a set of mechanisms that can reproduce it on demand.

Further, building on the idea of a virtual data grid, we view data regeneration as a special case of

replication. When a computational model executes, it generates some output data that may later become input data of some other models. The data may be replicated to many different locations for various reasons. Later on when the data are not needed, they may be removed. In the end, this would result in complete data removal. When a reader application reads from this data, data regeneration is needed since all copies of the data are gone. We call this the *Grid Data Life Cycle (GDLC)*, shown in Figure 1, from non-existence of data, to the original copy of new data (computed), to many copies of the data (replicated), to non-existence of data (deleted), and possibly back to one copy of the data (re-computation).

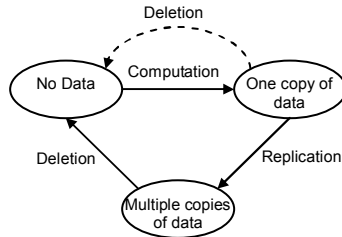


Figure 1 – Grid Data Life Cycle (GDLC)

We have developed a prototype system that manages the GDLC of the computational models and workflows in a platform neutral environment. This system, called *Active Data*, provides mechanisms to existing applications and workflows to run on the Grid. It also allows them to access data stored in different replica management systems, to associate metadata that describes how the data is computed across multiple replica systems, to ensure that data cannot be removed unless sufficient metadata for regeneration is associated, and to regenerate the required data transparently when needed during execution. Importantly, because *Active Data* is built under our GriddLeS system [6], no source code modification is needed.

2. Data Management Middleware

High performance computing applications in domains such as high energy physics, climate modelling and astronomy are often data intensive. They require access to large amounts of data, which are often copied and distributed in different locations in the Grid in order to improve reliability and system performance. While the Grid [14] describes software infrastructure that links and shares computational resources such as machines, scientific instruments and data, the term Data Grid [2] usually refers to a network of distributed storage resources that allows sharing of data distributed across remote repositories. Typically, these data can be accessed (e.g. file IO) transparently as if they were

stored locally. Today, many scientific communities are using Data Grids for managing and sharing large-scale distributed data, such as [32][33][34][35][37].

Distributed data (replicas) management on Data Grids has attracted a great deal of interests (e.g. [10][16][30]). A data grid management system must tackle issues such as user authentication, access control, storage resource management, scalability, performance and data uniqueness. Further, support for metadata management is important because researchers often associate metadata with scientific datasets, and this describes the contents and structure of the associated datasets. Examples of such systems include SRB [5], Globus RLS [22] and Gfarm [23].

Developed at the San Diego Supercomputer Centre, the Storage Resource Broker (SRB) is a data grid management system that employs a client-server architecture. It supports distributed data management using global logical namespace, including data publication, replication, transfer, discovery and preservation. The SRB uses logical namespaces to identify files, storage resources, users, metadata and access controls. Because these are done independently of the physical storage systems, a logical name that maps to a file does not change even if the file is moved to a different storage system. SRB supports various file systems and provides comprehensive client commands and APIs.

When an SRB object is registered, the SRB associates system-level metadata, such as location, size and creation date, with this object. For application-level metadata, SRB users are provided comprehensive metadata support via Scommanads and APIs. User-defined type of metadata can also be specified by using extensible schemas, which allows new tables to be added to the MCAT.

The Globus Toolkit [36] provides a number of components for supporting data management in the Grid. Specifically, the toolkit provides tools for data transfer (GridFTP and RFT) and replication (RLS and DRS). GridFTP [4] extends the standard FTP protocol to allow data to be transferred efficiently among remote sites. It is secure and uses multiple TCP streams for fast performance. However, one shortcoming is that when a client fails, GridFTP does not know where to restart the transfer because all the transfer information is stored in the memory. This means that a manual restart for data transfer is often required. To overcome this, the Reliable File Transfer (RFT) service [24] was developed. This non-user-based service is built on top of the GridFTP libraries and stores the transfer requests in a database rather than in memory. Clients are only required to submit a transfer request to the service and do not need to stay active because data transfer is managed by RFT on

behalf of the user. When an error is detected, RFT restarts the transfer from the last check point. The transfer status can be retrieved at anytime and RFT can also notify users when a requested transfer is complete.

Data replication is supported by the Globus Replica Location Service (RLS) [22] and, currently as a technical preview, the Data Replication Service (DRS) [3]. The RLS is an implementation of the RLS Framework [1]. It is a distributed registry that stores mapping information from unique logical identifiers to physical file locations. Using this information, RLS keeps track of the data replicas in the Grid. Client libraries and command line tools are provided for the user to register file copies, as well as to discover replica locations by querying the RLS server.

The RLS employs a metadata service, which contains information that describes the registered logical files. Descriptive data is attached to a file by creating and associating *attributes*. Currently, RLS supports values including decimal numbers, integers, strings, date and time. Attributes can be created using the provided client command. C and Java APIs are also available.

DRS is a high level data management tool that uses RLS and RFT. The main function of the DRS is to ensure files (replicas) exist on desired storage resources. This is done in three phases. The DRS firstly locates the desired files on the Grid by querying the available RLS servers. When the files, or replicas, are found, the DRS creates and submits a transfer request to RFT for copying the files to the desired storage resources. Finally, upon completion of the transfer, DRS registers the newly replicated files with the RLS servers.

Gfarm [23] also provides support for data replication and employs a metadata system to manage the file distribution. It focuses on building a distributed grid file system that provides scalable and high performance IO by integrating many local file systems and clusters. However, different to the SRB and the Globus Toolkit, the Gfarm metadata system is used to store file and parallel process information and currently does not provide support for users to associate descriptive data to the files.

The above systems all share the same goal; to provide mechanisms to manage data that are distributed geographically. However, in GDLC we also require high level tools that manage data access within workflows, data deletion when desired, and importantly, regeneration of the deleted data.

Data regeneration is a significant feature in a virtual data system, for example, as demonstrated in Chimera [15]. Chimera tracks how data is derived and allows deleted data to be recreated when needed. It uses the virtual data language (VDL) to describe

procedures that generate data and records information on how these procedures may be rerun later. This information is specified as VDL definitions, which are stored in the virtual data catalog (VDC). The corresponding data definitions are used when a workflow being created requires the data. Chimera creates an abstract workflow using the data definitions and stores it as an XML document. This document is then sent to the planner that generates executable forms of a workflow. Currently, Pegasus [11] is used by Chimera as the primary planner that converts the abstract workflow to a Condor DAGman DAG, a workflow that can be executed over the Grid resources.

One limitation of Chimera is that it does not provide support for remote IO operations. It copies the whole file from remote host to the execution node after a derivation of the file is done. Whilst this is efficient when the file size is small, it is inefficient when a reader application only requires a small portion of a large dataset.

Apart from data storage and regeneration, one key aspect in the GDLC is data removal. The storage systems mentioned above all provide ways to remove data from distributed resources. If a file is no longer required, a user may decide to delete it permanently. However, when a file is used by a computational model, the user may not want to delete it unless there is a way to recreate the file. Chimera does not provide facilities for the user to remove derived data and execution will fail if no derivation definition is defined for the deleted data.

3. GriddLeS and Kepler workflows

Grid workflows refer to workflows that are executed on the Grid. Such workflows are common in many domains, such as physics, earth sciences and astronomy, and often involve intensive computation and generate large amount of derived data. These data can be in various forms, such as datasets and images, and may serve as inputs to subsequent models.

Several workflow engines exist (e.g. [13][25][29]) that allow users to compose virtual applications using existing components. In this model, each component typically takes inputs and produces outputs that would become inputs of another component. Components within a workflow usually communicate by passing files or using interprocess communication pipes. When files are used, it is often difficult to overlap processes because a reader application needs to wait until the writer application finishes writing the output file. Also, when different machines are used for each process, the output file needs to be copied. To overcome this, we developed GriddLeS [6] and have also recently integrated it to the Kepler workflow system [13].

3.1 GriddLeS

GriddLeS is a middleware layer that provides a set of interprocess communication facilities for constructing arbitrary Grid applications/workflows from existing software components [6]. Such components are connected by a mechanism called GridFiles. A GridFile is a logical file that contains user-specified information about the physical file, including, but not limited to, resource location, file path, type of storage systems and user access information. GriddLeS employs a flexible architecture for supporting different file access mechanisms, such as local file access, remote file access and replica access. Importantly, no source code modification is required to access the GriddLeS run time, and this allows legacy codes to be run using a variety of different file access mechanisms. Previous work shows that Grid workflows can easily be constructed using legacy programs (e.g. Fortran programs) and performance can be significantly improved by pipelining and overlapping workflow components in parallel processes [8][27].

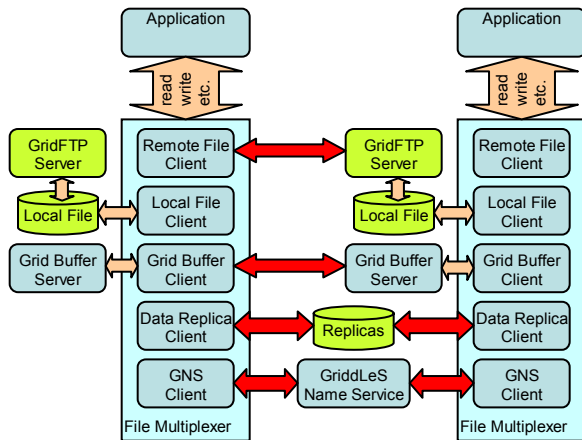


Figure 2 – The GriddLeS Architecture

GridFiles are supported by a device called the File Multiplexer (FM), which traps an application's IO system calls (e.g. open, close, read, etc.) and maps these operations dynamically to appropriate services. This IO interception mechanism enables applications to access files on the Grid without changing any code. Currently, GriddLeS provides access to local files, remote files, remote buffer for streaming, and data replicas in a number of different replica systems.

Instead of modifying the application, users need to specify the file mapping information in the GriddLeS Name Service (GNS). Each mapping in the GNS contains the original file path and the new file information. This is the central place for the users to tell GriddLeS what to do when a program is trying to

access a particular file. When file path changes, a user modifies the mapping information instead of the application code.

3.2 Kepler

Recently, we have developed a number of GriddLeS actors in Kepler that allow users to specify file mapping information directly into a Kepler workflow without using the GNS [7].

Kepler is an active open source project to build and evolve a scientific workflow system [13]. This system allows scientists from multiple domains to design and execute complex scientific workflows. Kepler is built on the Ptolemy II system [26], which provides a set of Java APIs for modelling heterogeneous, concurrent and hierarchical components using various models of computations (MoCs), which govern the interactions between software components. The Ptolemy II project studies modelling, simulation and design of concurrent, real-time, embedded systems.

In Kepler, an *actor* is a computational unit. It is a reusable component in a workflow that communicates via input and output ports. The interaction styles of actors are defined by *directors*, which are responsible for implementing particular MoCs. Examples of directors are Process Network (PN) and Synchronous Data flow (SDF). The former runs actors concurrently and the latter runs actors sequentially. The workflows are controlled by the directors. When changes are required (e.g. scheduling), only the director needs to be changed and no change to the workflow graph is required.

4. Active Data

The main goal of *Active Data* (AD) is to provide an infrastructure that allows the users to manage derived data within the GDLC. Specifically, such infrastructure must provide means for the users to:

- access data stored in different storage systems;
- enhance the performance with optimized data access;
- create data replicas when needed (and access the *best* one during runtime);
- create metadata that describes how the data may be recreated;
- delete the derived data (and ensure sufficient information on data regeneration is defined);
- regenerate the data when needed; and
- avoid source code modification in existing application components.

To support these, AD integrates various existing middleware, including GriddLeS and Kepler.

4.1 Data Access

In AD, data access is supported by GriddLeS. Conventionally, the user composes workflows by specifying file mapping information in the GNS. Each component within the workflow may access local files, remote files or replicated files. Streaming between two components can also be configured by using a location-independent buffer. This means that reader and writer applications can be overlapped using a pipe in the Grid for faster performance.

Using Kepler, the workflow can also be created without using the GNS. The user simply draws a workflow graph in Kepler with the desire director (e.g. a PN or a SDF director) and the GriddLeS actors. The file mapping information is specified within each actor by connecting the output port of an actor to the input port of another actor. Users can instruct GriddLeS to copy the output files to the execution node of the next actor via GridFTP or scp. Streaming is used when PN director is in use.

Each component (i.e. actor) can be executed in different resources and user authentication is done via Globus or SSH. When PN director is used, all actors are run concurrently and a buffer will be created between each pair of actors. In this mode, the components within the workflow are connected using pipes. On the other hand, when SDF director is used, Kepler will run the actors in a sequential order and third-party file transfer will be performed at the end of each actor execution.

4.2 Data Replication and Selection

GriddLeS employs a flexible plug-in architecture for supporting different file access mechanisms for various replica management systems [28]. It provides applications transparent access to the replicated data, in which more than one copy may exist. Each supported system provides means for users to replicate data and the replica information (e.g. number of replicas and their locations, creation dates, access level and user-defined metadata) is maintained by the storage systems. Such information is useful because GriddLeS will choose the best replica during program execution based on network statistics. It can also automatically switch to a better one dynamically in runtime.

4.3 Data Deletion and Metadata

AD provides a special client command (*adrm*) for deleting replicated files. Based on the GriddLeS' unified data access framework, AD employs an abstract interface for data removal from supported systems. The users may want to specify the physical file path (i.e. replica system-specific file information)

or the logical filenames that are used in the GNS. The latter is useful when the GNS is used for composing the workflows.

The major reason for this interface is to ensure that data regeneration information exists. Data can be removed permanently only when 1) there are replicas; or 2) data regeneration information associated as metadata. When the remove command is executed, it checks the existence of the file in the corresponding replica system. If there are more than one copy, it displays a list of the replicas and asks which one to be deleted. Then the command removes the chosen one permanently from the system. If the user wants to delete all replicas (or the last copy), it firstly checks whether there is data regeneration metadata associated, in which case it removes all the data but keeps the metadata in the system. An empty file still exists in the system because it is where the metadata will be associated with.

When no data regeneration metadata is available, the remove command requires the user to specify information about how data can be regenerated. Currently, AD supports regenerating data via running an arbitrary application (e.g. a linux binary), or a Kepler workflow. For an application, AD requires metadata such as physical file path to the binary, the arguments, and in the case of an application that outputs more than one file, which one is desired (Table 1). Input files for this application may be missing and require regeneration as well. If a Kepler workflow is used, then the user must save the workflow into a Kepler-specific XML document and store the document, and its path, in the replica system (Table 2).

Attribute	Value	Description
wf	no	no workflow is used
host	argos	where the program is
cmd	\$HOME/ccam	the program command
arg	> output.dat	command arguments
ofile	output.dat	the output file desired

Table 1 – Metadata information for arbitrary programs

Attribute	Value	Description
wf	kepler:/argos/poll.xml	the workflow

Table 2 – Metadata information for a workflow

Clearly, a Kepler workflow is preferred over a single binary because it allows program execution on Grid resources and all execution-related information (e.g. user authentication, input/output files, program paths, resources, etc.) is stored within the workflow. With Kepler's data provenance model [12], tracking and rerunning the derivation process of the data can be performed in more complex applications. Furthermore, regeneration information is stored in the replica system

as metadata and is not specific to a particular workflow system. Hence, it is possible to support other systems, including Chimera.

One problem of this design is that not all replica systems support user-defined (application-level) metadata. For example, SRB and Globus RLS both support this but Gfarm does not. Currently, Active Data overcomes this problem by storing the information in a simple text file. This file is registered in the Gfarm system so that it is accessible via the Gfarm native interface.

4.4 Data Regeneration on demand

AD provides facilities to regenerate deleted data on demand. During runtime, when a user-specified input file is not available, AD looks for the associated metadata. If there is no metadata, the execution will be terminated because an input file is missing. When data regeneration metadata exists, AD will try to rerun the procedures that created the data previously. If a Kepler workflow is used, AD firstly retrieves the associated XML document from the replica system and uses the Kepler client command interface to execute the workflow.

The data regeneration process is transparent to the reader application. AD blocks the reader application and reruns the previous application or workflow using the information from the metadata. If the executions are on the same host, then the reader application simply reads from the local file using GriddLeS once it is created. If an execution is performed on a remote resource, then the file may either be copied, or may be accessed remotely using GriddleS' remote IO operations. Further, rather than blocking the reader application, it is possible to stream the data between the writer and reader applications using the GriddLeS Buffer Service.

5. Evaluation and Results

A few case studies have previously reported the effectiveness of the GriddLeS data access mechanisms [27][28], and demonstrated that execution performance can be improved significantly using the data source optimization. A recent case study [7] was performed using a scientific workflow that computes the air pollution scenarios from a chain of atmospheric software components. The case study shows that it is easy to compose Grid workflows in the Kepler environment using existing code. In this example, Kepler was used to create, execute and monitor the entire virtual application. The result shows that the total execution time of the five computation actors almost reduced in a half when run in parallel.

Here we conducted a case study that considers some components of the pollution model workflow, including the global climate model (CCAM) [18], the data conversion program (cc2lam), the regional weather model (DARLAM) [19], another data conversion program (lam2cit) and the photo-chemical pollution model (CIT) [20]. The main aim of this case study is to demonstrate that a deleted file can be regenerated on the fly without significant performance loss. For evaluation purpose, two experiments were performed and all data and applications were run in the same machine.

5.1 Experiment 1 – Rerun an arbitrary application

The first experiment considers a writer application (CCAM) and a reader application (cc2lam). CCAM writes climate data to the file, *globaus8xx.197901*, and this serves as input to the conversion utility, cc2lam. During the experiment, CCAM was run and the output file was stored in an SRB server. The regeneration metadata were associated with the file and this was then subsequently deleted using AD's remove command. File mapping information was entered (to the GNS). When cc2lam was executed, AD found out that the input file was removed from the SRB server. It suspended the cc2lam execution, retrieved the associated metadata and used the information to rerun CCAM. The output data of CCAM was written to the original SRB file. The experiment was repeated 10 times to get an average execution times (and standard deviations) for each model.

The results are shown in Table 3. Originally the total execution time for the two models took a bit more than 30 minutes. With the data regeneration and SRB in place, the execution took roughly 2 more minutes to complete. Such loss is expected due to the overheads in GriddLeS, and the re-computation layer.

Model	Original	Std. Dev.	Active Data	Std. Dev.
CCAM	24:43	0.9	-	-
cc2lam	6:02	1.1	-	-
Total	30:45	1.3	32.37	1.2

Table 3 – Experiment 1 results

5.2 Experiment 2 – Rerun a Kepler workflow

The second experiment concerns rerunning a Kepler workflow using the native Kepler client command. A workflow that outputs fine-grained regional weather data for an air pollution model was created. Specifically, this workflow includes these components: CCAM, cc2lam, DARLAM and lam2cit, as shown in

Figure 3. The final output file (*winds_progfc_6km.bin*) serves as the input for the CIT program. Two workflows were actually created, one using SDF director and the other one using PN director.

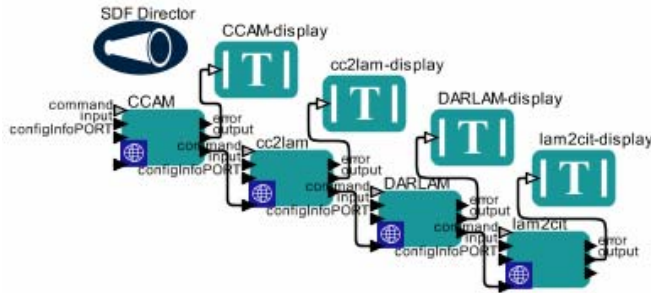


Figure 3 – Regional weather workflow using SDF

Two separate tests were performed and each test was repeated 10 times. The first test used the SDF workflow and the second one used the PN workflow. For each test, the output data of the workflow was written to the SRB server. After the execution, the workflow was saved into the Kepler XML file and this file was stored in the SRB server. The path to this workflow file was also saved as metadata in SRB. The output file was then deleted using the *adrm* command. After that, the CIT model, which requires data from the deleted file, was run.

In runtime, AD discovered that the file, *winds_progfc_6km.bin*, was missing. It then suspended CIT and re-executed the Kepler workflow using the XML document from the SRB server. Results are shown in Table 4. Rerunning the Kepler workflows took a few more minutes because of the initialization process of the Kepler environment before the workflow was run.

App.	Original (SDF)	Active Data (SDF)	Original (PN)	Active Data (PN)
Climate Model	46:01	-	39:24	-
CIT	30:24	-	14:21	-
Total	76:25	80:12	53:45	56:43

Table 4 – Experiment 2 results

6. Conclusion

This paper discusses Active Data, an infrastructure that combines existing middleware, including GriddLeS, Kepler and several replica systems, to support the GDLC. It provides support for data access, replication, selection, deletion and regeneration. Importantly, no source code modification is required and almost all existing applications can be supported.

The GriddLeS library is used extensively in Active Data because it supports legacy software components. It also employs a flexible architecture that provides applications transparent access to data in the Grid. The support for data streaming can also significantly improve the program performance. Besides, Active Data extends the virtual data concept to support data removal, where data files within workflows should not be deleted unless the system knows how to recreate them when required. Also, data generation information is stored as metadata and is not specific to any workflow systems. We plan to support more systems in the future.

Acknowledgements

The GriddLeS project is supported by the Australian Research Council and Hewlett Packard under an ARC Linkage grant, the Australian Government Department of Communications, Information Technology and the Arts under a GrangetNet grant, and the DART project. <http://dart.edu.au/>.

References

- [1] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney, *Giggle: A Framework for Constructing Scalable Replica Location Services*, SC2002, Baltimore, 2002.
- [2] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, *The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets*, Journal of Network and Computer Applications, 23:187-200, 2001.
- [3] A. Chervenak, R. Schuler, C. Kesselman, S. Koranda, and B. Moe, *Wide Area Data Replication for Scientific Collaborations*, Proceedings of 6th IEEE/ACM International Workshop on Grid Computing (Grid2005), November 2005.
- [4] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke, *Data Management and Transfer in High Performance Computational Grid Environments*, Parallel Computing Journal, 28:749-771, May 2002.
- [5] C. Baru, R. Moore, A. Rajasekar and M. Wan, *The SDSC Storage Resource Broker*, Proceedings of CASCON'98 Conference, Toronto, Canada, 1998.
- [6] D. Abramson and J. Komineni, *A Flexible IO Scheme for Grid Workflows*, IPDPS-04, Santa Fe, New Mexico, April 2004.
- [7] D. Abramson, J. Kommineni and I. Altinas, *Flexible IO services in the Kepler Grid Workflow Tool*, Proceedings of 1st International Conference one-Science and Grid Computing, Melbourne, 2005.
- [8] D. Abramson, J. Komineni, J. McGregor and J. Katzfey, *An Atmospheric Sciences Workflow and its*

- Implementation with Web Services*, Future Generation Computer Systems, 21:69-78, 2005.
- [9] D. Bernholdt, S. Bharathi, D. Brown, K. Chancio, M. Chen, A. Chervenak, L. Cinquini, B. Drach, I. Foster, P. Fox, J. Garcia, C. Kesselman, R. Markel, D. Middleton, V. Nefedova, L. Pouchard, A. Shoshani, A. Sim, G. Strand and D. Williams, *The Earth System Grid: Supporting the Next Generation of Climate Modeling Research*, In Proceedings of the IEEE, 93(3):485-495, March, 2005.
 - [10] D. Duellmann, W. Hoschek, J. Jean-Martinez, A. Samar and B. Segal, *Models for Replica Synchronization and Consistency in Data Grid*, 10th IEEE Symposium on High Performance and Distributed Computing (HPDC-10), 2001.
 - [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.
 - [12] I. Altintas, O. Barney, and E. Jaeger-Frank, *Provenance Collection Support in the Kepler Scientific Workflow System*, IPAW2006, Chicago, 2006.
 - [13] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, *Kepler: An Extensible System for Design and Execution of Scientific Workflows*, SSDBM'04, Santorini Island, Greece, June 2004.
 - [14] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.
 - [15] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, *Chimera: A Virtual Data System for Representing, Querying and Automating Data Derivation*, In 14th Conference on Scientific and Statistical Database Management, Edinburgh, Scotland, July 2002.
 - [16] I. Foster, J. Voeckler, M. Wilde and Y. Zhao, *The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration*, Proceedings of the CIDR 2003 First Biennial Conference on Innovative Data Systems Research, January 2003.
 - [17] J. Gray, A. S. Szalay, A. R. Thakar, C. Stoughton, and J. vandenBerg, *Online Scientific Data Curation, Publication, and Archiving*, Proceedings of SPIE, Virtual Observatories, 4846:103-107, Dec 2002.
 - [18] J. L. McGregor, K. C. Nguyen and J. J. Katzfey, *Regional Climate Simulations using a Stretched-grid Global Model*, Research Activities in Atmospheric and Oceanic Modeling, H. Ritchie (ed.), pp. 3.15-3.16, 2002.
 - [19] J. L. McGregor, K. J. Walsh and J. J. Katzfey, *Nested Modelling for Regional Climate Studies*, A. J. Jakeman, M. B. Beck and M. J. McAleer (eds.), Modelling Change in Environment Systems, J. Wiley and Sons, pp. 367-386, 1993.
 - [20] K. J. Tory, M. E. Cope, G. D. Hess, S. H. Lee and N. Wong, *The use of long-range transport simulations to verify the Australian Air Quality Forecasting System*, Australian Meteorological Magazine, 52(4):229-240, 2002.
 - [21] L. Bright and D. Maier, *Deriving and Managing Data Products in an Environmental Observation and Forecasting System*, In Proceedings of Conference on Innovative Data Systems Research (CIDR), 2005.
 - [22] M. Manohar, A. Chervenak, B. Clifford and C. Kesselman, *A Replica Location Grid Service Implementation*, Data Area Workshop, Global Grid Forum 10, 2004.
 - [23] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, H. Sato, Y. Tanaka, S. Sekiguchi, Y. Watase, M. Imori, and T. Kobayashi, *Worldwide Grid Data Farm for Petascale Data Intensive Computing*, Technical Report, Electro-technical Laboratory, ETL-TR2001-4, 2001.
 - [24] R. Madduri, C. Hood, W. Allcock, *Reliable File Transfers in Grid Environments*, Proceedings of the 27th IEEE Conference on Local Computer Networks, 2002.
 - [25] S. Majithia, M. S. Shields, I. J. Taylor, and I. Wang, *Triana: A Graphical Web Service Composition and Execution Toolkit*, Proceedings of the IEEE International Conference on Web Services (ICWS'04), 2004.
 - [26] S. S. Bhattacharyya, E. Cheong, J. Davis II, M. Goel, C. Hylands, B. Kienhuis, E. A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie, N. Smyth, J. Tsay, B. Vogel, W. Williams, Y. Xiong, and H. Zheng, *Heterogeneous Concurrent Modeling and Design in Java*, Technical Memorandum UCB/ERL M02/23, EECS, University of California, Berkeley, August 5, 2002.
 - [27] T. Ho and D. Abramson, *A GriddLeS Data Replication Service*, Proceedings of 1st International Conference one-Science and Grid Computing, Melbourne, 2005.
 - [28] T. Ho and D. Abramson, *A Unified Data Grid Replication Framework*, Proceedings of 2st International Conference one-Science and Grid Computing, Amsterdam, 2006.
 - [29] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat and P. Li. *Taverna: A tool for the composition and enactment of bioinformatics workflows*, Bioinformatics Journal, 20(17):3045-3054, 2004.
 - [30] W. Allcock, A. Chervenak, I. Foster, C. Kesselman, C. Salisbury and S. Tuecke, *The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets*, Journal of Network and Computer Applications, 23:187-200, 2001.
 - [31] Y. L. Simmhan, B. Plale, D. Gannon, *A Survey of Data Provenance in e-Science*, SIGMOD Record, 34(3):31-36, 2005.
 - [32] BIRN: Biomedical Informatics Research Network. <http://www.nbim.net>.
 - [33] National Virtual Observatory. <http://www.srl.caltech.edu/nvo/>.
 - [34] Network for Earthquake Engineering Simulation. <http://www.eng.nsf.gov/nees/>.
 - [35] ROADNet: Real-time Observatories, Applications and Data Management. <http://roadnet.ucsd.edu>.
 - [36] The Globus Toolkit. <http://www.globus.org/toolkit/>.
 - [37] The Grid Physics Network. <http://www.griphyn.org/proj-desc1.0.html>