

Flexible IO Services in the Kepler Grid Workflow System

§ David Abramson, § Jagan Kommineni, † Ilkay Altintas
{davida,jagan@csse.monash.edu.au, altintas@sdsc.edu}

§ School of Computer Science and Software Engineering, Monash University, Clayton, 3800, Victoria, Australia

† San Diego Supercomputing Center, 9500 Gilman Drive, MC 0505 La Jolla, CA 92093-0505, USA.

Abstract

Existing Grid workflow tools assume that individual components either communicate by passing files from one application to another, or are explicitly linked using interprocess communication pipes. When files are used it is usually difficult to overlap reader and write execution. On the other hand, interprocess communication primitives are invasive and require substantial modification to existing code. We have built a library, called GriddLeS, that combines the advantages of both approaches without any changes to the application code. GriddLeS overloads conventional file IO operations, and supports either local, remote or replicated file access, or direct communication over pipes. In this paper we discuss how GriddLeS can be combined with workflow packages and show how flexible and powerful virtual applications can be constructed rapidly. A large atmospheric science case study is discussed.

1 Introduction

Grid computing has been proposed as the next generation of infrastructure to support distributed applications in science, engineering and business [1][2][3][4]. The Grid provides mechanisms that harness computational resources, databases, high speed networks and scientific instruments, allowing users to build innovative *virtual applications*. Such virtual applications are synthesized by combining multiple different components on multiple computational resources.

One of the more significant challenges for Grid computing remains the difficulty in developing software that is both concurrent and distributed. Whilst there have been many proposals over the years on how to build such systems, including very recent work in Web services [5][6][7], programming still remains a low level and error prone task.

Recently, a number of different groups have developed workflow solutions that double as programming environments for the Grid. These *Grid Workflow* systems[8][9][10][11][12][13][14][15] allow a user to compose a complex virtual application based on pre-existing, in some case, legacy components. In this model, components typically take input and produce output as part of a pipeline. The workflow system schedules the computations on the most appropriate (or selected) resource only when the inputs are available. Likewise, when the output is produced, it is forwarded to the next computation in the pipeline. Grid Workflows have been applied to diverse fields such as Computational Chemistry[10], Ecology[11] and bio informatics [12].

Existing Grid workflow tools assume that individual components either communicate by passing files from one application to another, or are explicitly linked using interprocess communication pipes. When files are used, it is usually difficult to overlap reader and write execution because the file is only copied from one computation to the next once it has been completed and closed. This makes it difficult to develop deeply pipelined computations in which input is consumed at the same time as it is produced. Accordingly, pipelined computations usually require the use of interprocess communication primitives such as TCP/IP pipes. However, these are usually invasive and require substantial modification to existing code. Moreover, once coded, it is difficult to revert to use a more conventional file-based IO mechanism.

We have built a library, called GriddLeS, that combines the advantages of both approaches without any changes to the application code [16]. GriddLeS overloads conventional file IO operations, and support either local, remote or replicated file access, or direct communication over pipes. Thus, the program thinks it is reading or writing a file, but GriddLeS may choose an alternative transport at run time. This choice of transport is made late in the configuration process,

providing significant flexibility in building workflows from pre-existing stand alone applications.

Previously we have reported success at using legacy components as part of larger Grid workflows [17][18]. The most noticeable advantage of our work is that the legacy codes operated in the Grid workflow without any source modification – even when the computation was deeply pipelined as discussed. Because the system is flexible, it is possible to use an application as part of a pipelined workflow, but then resort to running it as a conventional file-based application on another occasion. For each of the case studies reported, we hand scheduled, composed and executed the applications. In this paper we discuss how GriddLeS can be combined with existing Grid workflow packages such Kepler, a public domain grid workflow system [8]. We illustrate how flexible and powerful virtual applications can be constructed rapidly. A case study of building a large atmospheric science application is discussed.

2 Grid Programming in Kepler

Kepler is an active open source cross-project, cross-institution collaboration to build and evolve a scientific workflow system on top of the (also evolving) Ptolemy II system. The emerging Kepler system allows scientists from multiple domains (bioinformatics, cheminformatics,ecoinformatics, geoinformatics, astrophysics etc.) to design and execute scientific workflows. Scientific workflows can be used to combine data integration, analysis, and visualization steps into larger, automated "scientific process pipelines" and "grid workflows" [9][10].

Kepler builds upon the mature Ptolemy II framework [19][20], developed at the University of California, Berkeley. Ptolemy II is a Java-based software framework with a graphical user interface called Vergil. The Ptolemy II project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interactions between components. Along with a set of APIs for modeling behaviour, it can be used for heterogeneous hierarchical modeling [20].

The focus of the Ptolemy II system is to build models of systems based on the assembly of pre-designed components. These components are called actors [19]:

“An actor is an encapsulation of parameterized actions performed on input data to produce output data. An actor may be state-less or state-full, depending on whether it has internal state. Input and output data are

communicated through well-defined ports. Ports and parameters are the interfaces of an actor. A port, unlike methods in Object-Oriented designs, does not have to have call-return semantics. The behaviors of a set of actors are not well-defined without a coordination model. A framework is an environment that actors reside in, and defines the interaction among actors.”

The interaction styles of actors are captured by models of computation (MoC). A MoC defines the communication semantics among ports and the flow of control and data among actors. There are two modes of pipelining in Kepler, namely files or messages.

A framework implements a model of computation. Frameworks and actors together define a system [19]. In our Grid Workflow framework, we define a set of grid actors in Kepler that work in dataflow-based computation models such as Process Network (PN) and Synchronous Data Flow (SDF). Directors are responsible for implementing particular MoCs, and thus they define the “orchestration semantics” of the workflow. Simply by changing the director of a workflow, one can change the scheduling and overall execution semantics of a workflow, without changing any of the components or the network topology of the workflow graph.

The Process Networks (PN) and Synchronous Data Flow (SDF) directors which we will mention in this paper are based on Kahn Process Networks and dataflow. A process network is a directed graph, comprising a set of nodes (processes) connected by a set of directed arcs (representing FIFO queues). Each process executes as a standalone sequential program and is wrapped as a Ptolemy II actor. The one-way FIFO channels are used for the communication of processes and each channel can carry a possibly infinite sequence (a stream) of atomic data objects (tokens). Since channels have in principle unbounded capacity, writes to channels are non-blocking, while reads are blocking [19]. The SDF domain is a dataflow-based execution model in which a sequential execution order of actors can be statically determined prior to execution. This results in execution with minimal overhead, as well as bounded memory usage and a guarantee that deadlock will never occur..

Kepler extends PtolemyII with a significant number of actors aimed particularly at scientific applications, e.g., for remote data and metadata access, data transformations, data analysis, interfacing with legacy applications, web service invocation and deployment, provenance tracking, etc. Target application areas include bioinformatics, cheminformatics, ecoinformatics, and geoinformatics workflows among others. Kepler also inherits from Ptolemy the actor-oriented modeling paradigm. There is research and development to build new computation

models (and so directors) targeted for scientific applications as well. Through actor-oriented and hierarchical modeling features built into Ptolemy, Kepler scientific workflows can operate at very different level of granularity, from low-level "plumbing workflows" that explicitly move data around, start and monitor remote jobs, etc. to high-level "conceptual workflows" that interlink complex, domain specific data analysis steps.

Grid workflows (and distributed applications in general) can be seen as special scientific workflows. They involve high performance and/or high throughput computational tasks. Much work in grid workflows has focused on improving application performance through schedulers that optimize the use of computational resources and bandwidth. One of our targets to run distributed applications in Kepler was a framework that supports the design and reuse of grid workflows. Individual workflow components (e.g., for data movement, database querying, job scheduling, remote execution etc.) are abstracted into a set of generic, reusable tasks. [19] Our goals for a distributed Kepler include also the ability to easily form ad-hoc networks of cooperating Kepler instances. Each Kepler cooperating network (collaboration) can have access constraints and allows Kepler models or submodels to be run on participating instances. Once a Kepler cooperating network has been created, it can configure one or more subcomponents of a workflow to be distributed across nodes of the newly constructed network (Grid).

3 GriddLeS

GriddLeS is a library that provides a rich set of interprocess communication facilities. Such facilities can be used to implement data sharing in a Grid workflow. The core concept in GriddLeS is GridFiles, a device which enables file based interprocess communication between software components. GridFiles are supported by a component called the File Multiplexer (FM), as shown in Figure 1.

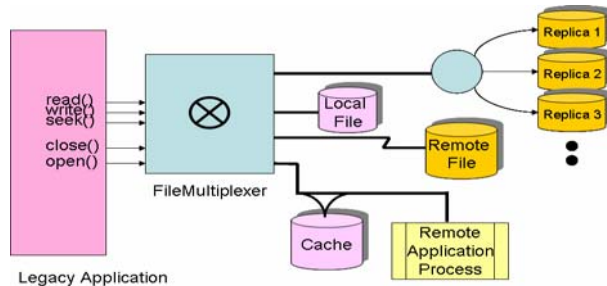


Figure 1 - The GriddLeS Architecture

The FM traps IO system calls from an application (e.g. open, close, read, write, etc.) and maps these operations to appropriate services dynamically. GriddLeS uses two different trapping mechanisms based on Bypass [21], and Parrot [22]. These are generic techniques that make it possible to intercept function calls on a variety of platforms and operating systems.

When a program reads data, the FM can redirect the read operation to read from a local file, a remote file, a replicated file or connect to a write operation on a local or remote machine using a shared buffer. These are processed by the Local File Client, the Remote File Client, the GRS Client, and the Grid Buffer Client respectively, as shown in Figure 2. These three access modes make it possible to build flexible Grid workflows in which computations either read and write their data to and from files, or communicate directly via pipes. The latter allows writer and reader application to overlap IO and computation, and leads to efficient solutions when stream based computations are performed.

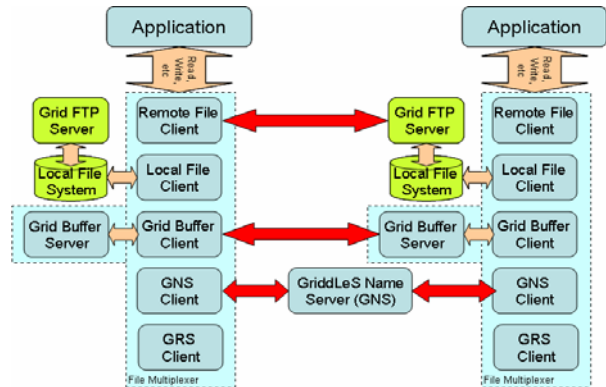


Figure 2 - Architecture of GriddLeS

The Local File Client is used to provide the running program access to local files. The Remote File Client provides access to remote files using a range of different file transfer mechanisms, such as GridFTP [23], SRB [24] or scp [25]. The GRS Client is used to implement a GriddLeS data Replication Service. When an application opens a replicated file, the GRS makes an actual binding to one of those replicas [24][26]. It determines the most appropriate one by measuring the available bandwidth to each replica (using tools such as the Network Weather Service (NWS) [27], and it dynamically switches source during program execution should the bandwidth change. The GRS has been designed to support a variety of replication services, but the current implementation uses the Storage Resource Broker (SRB) from SDSC [24] and the Globus Replica Location Service (RLS) [26].

The Grid Buffer Client is required for inter-process pipelined communication. On the writer's side, it connects and sends written data to a remote Grid Buffer Server. On the reader's side it reads the data from the buffer server. Data is stored by the Grid Buffer Service in a hash table rather than a sequential file to support random access. If the data read by the reader application has not been written when a read is requested, the reader blocks until the needed data is available. These additions make GridFiles significantly more powerful than conventional Unix pipes, that typically only operate on standard IO channels and also don't work across machine domains. The Grid Buffer Server is built using Web Services.

The GriddLeS Name Service (GNS) is used for storing file mapping information that describes the configuration of a particular Grid application. The mapping is basically a key-value pair, containing the server name and the full path to the target file. This is a central place for the user to tell GriddLeS what to do when a program is trying to access a particular file (e.g. during the Open call). During runtime, GriddLeS reads the information and stores the values to a hash table. The mapping information can be changed at any time, and GriddLeS updates the information dynamically. In our previous work the GNS has been loaded manually, based on the structure of a Grid workflow. In the next section we discuss how we have used Kepler to load the GNS.

4 GriddLeS within Kepler

As discussed in Section 2, users can build arbitrarily complex virtual applications by concatenating a series of complete application components. Moreover, components can be controlled by two different directors that affect the timing and behaviour of the system. Whilst it is possible to alter the choice of director without changing the workflow, the two directors discussed, namely SDF and PN, require significant changes to the underlying application. For example, if a SDF director is used, then the applications read and write files (as it is single threaded and there is no concurrency), and these files are copied from one resource to another. On the other hand, if a PN director is used, the applications all run concurrently (multi-threaded), and inter-process pipes are required. These two IO methods are normally different enough that the source code of the components must be altered. However, as discussed in Section 3, GriddLeS supports both file IO and inter-process communication by abstracting conventional IO primitives, and thus source modification is not required. Thus, combining GriddLeS and Kepler offers substantial new functionality.

Figure 3 shows a typical Kepler actor, representing a global climate model, a large legacy program that solves equations that describe the state of the atmosphere. The actor has 11 input ports and 11 output ports, and these correspond to 11 input files and 11 output files that the application reads and writes. In this example, some of the files contain specific time stepped data holding fields such as the temperature and pressure values distributed over the computational grid, whilst others hold status parameters and initial values. When the program is run on the command line, it expects the input files to be available, and creates and writes data to the output files. Normally, the input files are set up prior to a run, and are configured for a particular experiment. One file, however, contains simulation 'forcing' data, can either be sourced from a static file, or may be taken from the output of another computation. In this latter case, a Kepler workflow can be used to represent the dataflow between the models.

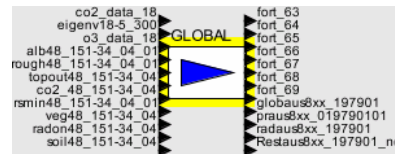


Figure 3 - Kepler actor representing a global climate model

Figure 4 shows a sample workflow in which a number of these atmospheric models are coupled. Importantly, most of the input ports are connected to static files that are previously created as part of the experimental setup, and thus are not shown on the workflow. The actual file names are entered into a dialogue box for each input port. However, the forcing data is taken from one application and piped directly into the next one in the workflow. In this example, a global climate model called CCAM sends data to a data conversion utility called cc2lam, which in turn sends forcing data to regional weather model called DARLAM. DARLAM in turn, sends data to another conversion program called lam2cit, which sends data to a pollution model called CIT. This sample workflow will be discussed in more detail in the next section.

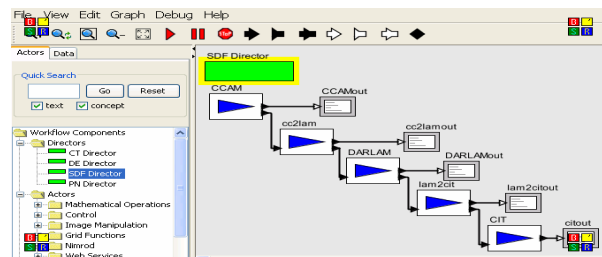


Figure 4 - Sample atmospheric sciences workflow

The example in Figure 4 shows the SDF director controlling the experiment. The choice of SDF means that the programs are run sequentially, and the next program in the workflow is not started until the previous one is complete. If the source and destination applications are run on the same machines, then the files are simply placed in the file system, and these are accessed by both source and destination computations. However, if the source and destination applications are run on different machines (which do not share a file system), then the data must be copied from one to the other before the second one can start. If the SDF director is swapped for the PN director, then all models (actors) start executing at once. The models used in this example solve a set of time stepped differential equations, and thus can emit time stepped fields as they are computed. Likewise, the forcing data can be consumed by a computation as it is read. Thus, if a PN director is specified, a downstream computation can start execution before the driving one is complete, and the workflow can be executed as a streaming pipeline. *Whilst these two different IO modes are available in the standard distribution of Kepler, by combining the GriddLeS IO library with Kepler, we are able to swap from one IO mode to the other without changes in the application source code.*

When used with Kepler, GriddLeS actually supports four different IO modes between components, as follows:

1. Output data is written to a local file which is copied from the source machine to the destination machine. The downstream computation reads the input data from a file. The experiment is controlled by the SDF director which means that a destination computation is only started when the source completes;
2. Output data is written to a remote file on the destination machine. The downstream computation then reads the input data from that file, and the experiment is controlled by the SDF director as in (1);
3. Output data is written to a local file on the source machine. The computation on the destination machine reads the data remotely and the experiment is controlled by the SDF director as in (1);
4. Output data is written to a buffer, and is read by the downstream computation from the same buffer. The experiment is controlled by the PN director which means that all computations run concurrently.

Each of these IO modes has different performance characteristics, and they need to be chosen based on the performance profile of both the applications and the machines on which they run. For example, it only makes sense to run the actors in parallel if they run for

a reasonable amount of time, sufficient resources are available, and they consume data as they produce it. Importantly, it is possible to switch between them with minor changes to the Kepler workflow configuration, for example, by changing the director or some attributes of the edges between ports. Equally importantly, none of the different configurations require any changes to the source of the applications. This means that a user can experiment with different IO configurations by interacting with the Kepler interface alone and selecting the configuration choices visually via available parameters.

GriddLeS actors for the applications, such as those depicted in Figure 4, are created using a special Kepler actor called the GridletCreator. The GridletCreator takes a specification for the application, which indicates the number of input and output ports, and how to invoke the program. This information is used to build a new actor that can be instantiated using the usual Kepler user interface. We have actually created two different GridletCreator actors – one for running the application on the local machine, and another for running it on a remote resource. We have implemented a number of different methods of invoking a remote application, ranging from a simple Web Service through to a Nimrod/G submission [4].

In section 3 we indicated that the GriddLeS Name Service (the GNS) contains the configuration information for each file. When used with Kepler, the GNS is initialized using the information in the Kepler workflow XML description. Thus, when the applications open their files, the behaviour of the IO calls is effectively controlled by the information in the actor parameters.

5 Performance Results

In this section we present the results of a real workflow designed to compute air pollution scenarios from a chain of atmospheric science codes as an example of what is possible. The experiment illustrates how easy it is to create a virtual application using the Kepler user interface. In this case the entire application is created, executed and monitored from within Kepler. The workflow is one that we have previously executed manually [17][18], and which was introduced in Section 4. Specifically, a global climate model (CCAM) is used to drive the boundaries of a region weather model (DARLAM). The output of the regional model drives the weather scenario in a photo-chemical pollution model (CIT). This chain of computations allows scientists to explore the effect of different weather scenarios on pollution outcomes, and has been applied many times in urban airshed studies [28][29].

Actors that represent the 3 models (CCAM, DARLAM and CIT), together with two conversion utilities (cc2lam and lam2cit), are created using the GridletCreator actor discussed in section 4. CCAM, DARLAM and CIT require significant computational resources and thus are run on different computer systems. However, cc2lam and lam2cit consume so little CPU time, that they can be executed on the same machines as the main models. After filling the necessary configuration information required by the GridletCreator, providing file names for the fixed ports and connecting the ports in which communication is required, the Kepler workflow model is saved and the GNS configuration information is updated.

The workflow can be executed by simply clicking the run button on the Kepler control window, as shown in Figure 4. In this experiment we use both SDF (Synchronous Data Flow director and PN (Process Network) director.

The testbed for the experiment is shown in Figure 5, and the mapping of models to servers is shown in Figure 6.

romulus	romulus.dstc.monash.edu.au	Intel P4, 2.4 GHz
variton	variton.csse.monash.edu.au	Intel P4, 2 GHz
brecca	brecca-2.vpac.org	Intel Xeon, 2.8 GHz, 124 node cluster

Figure 5 - Testbed details

Model	Execution Server	Result Server
CCAM	Romulus	Romulus
cc2lam	Romulus	Variton
DARLAM	Variton	Variton
Lam2cit	Variton	Brecca
CIT	Brecca	Brecca

Figure 6 - Mapping models to servers

Figure 7 shows the performance results of the experiment under the four different IO configurations discussed in Section 4. The execution time for each configuration is made up of the computation times for each of the 5 components. Configuration 1 writes the output files on the local server, and reads them from the local server. The files are copied from one server to the other using GridFTP. Configuration 2 writes the files onto the downstream server using a remote write function, and thus the reader can use a local read. Configuration 3 writes on the local server, but the downstream computation reads the file using a remote read operation. Configuration 4 uses overlapped IO via a buffer. Configuration 4 is the fastest because the sending and receiving programs are able to overlap writing, reading and computation. The next fastest is

configuration 1, in which the file is written and read locally, and a single efficient file transfer moves the data in one step. Configurations 2 and 3 are less efficient again because the writing or reading application must access data across the network. These results are consistent with our previous experiments [17][18], but in this context we are able to control the differences using Kepler.

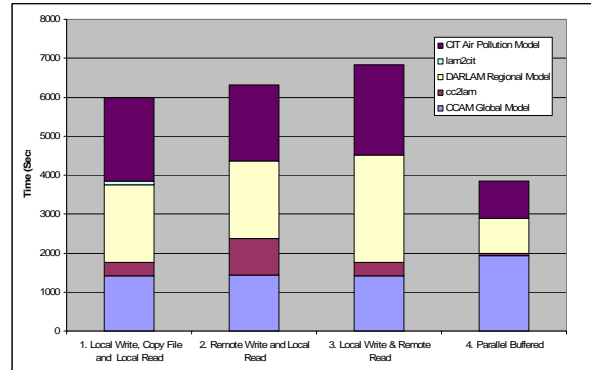


Figure 7 - Experimental Results

6 Conclusions

Together, Kepler and GriddLeS, provide a powerful implementation of Grid workflows. On its own, Kepler provides functionality to construct, edit and execute workflows using SDF or PN execution models. However, Kepler requires substantial modifications to the underlying applications in moving from one mode to another, making it difficult to tune the performance of the overall system. On the other hand, GriddLeS offers the ability to delay the choice of inter-process data transfer mechanism until run time. Since no application changes are required, a user can optimize and tune the performance of the system at run time using the same Kepler interface that they used to build the original workflow. Our initial experiments with Kepler are very positive.

In the future we plan to modify the SDF and PN directors so that interact with Grid scheduling systems such as Nimrod. We also plan to expose the GriddLeS replica service (GRS) so it can be used by Grid workflows.

Acknowledgements

The GriddLeS project is supported by Australian Research Council and Hewlett Packard under and ARC Linkage grant, and the Australian Government Department of Communications, Information Technology and the Arts under a GranetNet grant.

We would like to thank our colleagues, John McGregor, Jack Katzfey and Martin Dix from the CSIRO Division of Atmospheric Sciences who provided the models used in this experiment.

We are grateful to Kepler and Ptolemy development teams for their support. The related work was supported by NSF/ITR 0225676 (SEEK), DOE SciDAC DE-FC02-01ER25486 (SDM), NSF/ITR CCR-00225610 (Chess), NSF/ITR 0225673 (GEON), NSF/ITR 0325963 (ROADNet) and NSF/DBI-0078296 (Resurgence).

References

- [1] I. Foster and C. Kesselman, (editors), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.
- [2] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *Int'l J. of Supercomputer Applications*, vol. 11, no. 2, 1997, pp. 115-128.
- [3] The Globus Toolkit: <http://www-unix.globus.org/toolkit/>
- [4] D. Abramson, J. Giddy and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid," In *Int'l. Parallel and Distributed Processing Symposium (IPDPS)*, Cancun, Mexico, May 2000. <http://www.csse.monash.edu.au/~davida/nimrod/>.
- [5] R. van Engelen, "The gSOAP toolkit 2.0.," *Technical Report*, Florida State University, <http://www.cs.fsu.edu/~engelen/soap.html>, 2005.
- [6] Sun Microsystems, "Web Services Made Easier: The Java APIs and Architectures for XML", 2005, <http://java.sun.com/webservices/jwsdp/index.jsp>.
- [7] J. Kommineni, "Grid Applications: A Seamless Approach with Web Services," *The APAC Conference and Exhibition on Advanced Computing*, Grid Applications and eResearch Royal Pines Resort Gold Coast, Queensland Australia. 29 September - 2 October, 2003.
- [8] <http://kepler-project.org>
- [9] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao and Y. Zhao, "Scientific Workflow Management and the Kepler System", *Concurrency and Computation: Practice & Experience*, Special Issue on Scientific Workflows, 2005.
- [10] Ilkay Altintas, Adam Birnbaum, Kim Baldrige, Wibke Sudholt, Mark Miller, Celine Amoreira, Yohann Potier and Bertram Ludaescher, "A Framework for the Design and Reuse of Grid Workflows" *Intl. Workshop on Scientific Applications on Grid Computing (SAG'04)*, LNCS 3458, Springer, 2005.
- [11] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher and S. Mock, "Kepler: Towards a Grid-Enabled System for Scientific Workflows," in the *Workflow in Grid Systems Workshop in GGF10 - The 10th Global Grid Forum*, Berlin, March 2004.
- [12] Taverna Project: <http://taverna.sourceforge.net>
- [13] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *Technical Report GRIDS-TR-2005-1*, Grid Computing and Distributed Systems Laboratory, Univ of Melbourne, 2005. <http://www.gridbus.org/reports/GridWorkflowTaxonomy.pdf>.
- [14] e-Science Grid Environments Workshop, e-Science Institute, Edinburgh, Scotland, May 2004, <http://www.nesc.ac.uk/esi/events/>.
- [15] Scientific Data Management Framework Workshop, Argonne National Labs, August 2003. <http://sdm.lbl.gov/~arie/sdm/SDM.Framework.ws hp.htm>.
- [16] D. Abramson and J. Kommineni, "A Flexible IO Scheme for Grid Workflows," *IPDPS-04*, Santa Fe, New Mexico, April 2004.
- [17] D. Abramson, J. Kommineni, J.L. McGregor and J. Katzfey, "An Atmospheric Sciences Workflow and its Implementation with Web Services," *Future Generation Computer Systems, The Int'l J. of Grid Computing: Theory, Methods and Applications*, vol. 21, 2005, pp. 69-78.
- [18] J. Kommineni and D. Abramson, "Building Virtual Applications for the GRID with Legacy Components," *European Grid Conference*, held at Science Park Amsterdam, The Netherlands, February 14-6 2005.
- [19] <http://ptolemy.eecs.berkeley.edu/ptolemyII/ptIIfaq.htm>
- [20] C. Brooks, E.A. Lee, X. Liu, S. Neuendorffer, Y. Zhao and H. Zheng, "Heterogeneous Concurrent Modeling and Design in Java (Volumes 1-3)," *Technical Report*, Dept. of EECS, University of California, Berkeley, 2004.
- [21] D. Thain and M. Livny, "Bypass: A tool for building split execution systems," in *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing*, Pittsburgh, Pennsylvania, August 1-4, 2000, pp. 79-85.

- [22] D. Thain and M. Livny, "Parrot: Transparent User-Level Middleware for Data-Intensive Computing," *Workshop on Adaptive Grid Middleware*, New Orleans, Louisiana, September 2003.
- [23] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu and I. Foster, "The Globus Striped GridFTP Framework and Server," *Proceedings of Super Computing 2005 (SC05)*, November 2005.
- [24] A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S. Chen and R. Olschanowsky, "Storage Resource Broker - Managing Distributed Data in a Grid," *Computer Society of India Journal, Special Issue on SAN, Vol. 33, No. 4*, 2003, pp. 42-54.
- [25] <http://www.openssh.com/manual.html>
- [26] A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman and R. Schwartzkopf, "Performance and Scalability of a Replica Location Service," *Proceedings of the International IEEE Symposium on High Performance Distributed Computing (HPDC-13)*, June 2004.
- [27] B. Gaidioz, R. Wolski, and B. Tourancheau, "Synchronizing Network Probes to avoid Measurement Intrusiveness with the Network Weather Service," *Proceedings of 9th IEEE High-performance Distributed Computing Conference*, August, 2000, pp. 147-154.
- [28] J. L. McGregor, K. C. Nguyen and J. J. Katzfey, "Regional climate simulations using a stretched-grid global model. In: Research activities in atmospheric and oceanic modeling," H. Ritchie (ed.). (*CAS/JSC Working Group on Numerical Experimentation Report; 32; WMO/TD - no. 1105*) [Geneva]: WMO. 2002, pp.3.15-3.16.
- [29] 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. In: Modelling and predicting extreme events": extended abstracts of presentations at the fourteenth annual *BMRC Modelling Workshop*, BMRC, Melbourne, A. J. Hollis, and P. J. Meighen (editors) (BMRC Research Report, 90) Melbourne: Bureau of Meteorology Research Centre., 2002, pp. 19-23.