

The GriddLeS Data Replication Service

Tim Ho and David Abramson
{tim.ho, davida}@csse.monash.edu.au

School of Computer Science and Software Engineering
Monash University,
900 Dandenong Rd,
Caulfield East, 3145, Australia

Abstract

The Grid provides infrastructure that allows an arbitrary application to be executed on a range of different computational resources. When input files are very large, or when fault tolerance is important, the data may be replicated. Existing Grid data replication middleware suffers from two shortcomings. First, it typically requires modification to existing applications. Second, there is no automatic resource selection and a user must choose the replica manually to optimize the performance of the system. In this paper we discuss a middleware layer called the GriddLeS Replication Service (GRS) that sits above existing replication services, solving both of these shortcomings. Two case studies are presented that illustrate the effectiveness of the approach.

1 Introduction

The Grid provides infrastructure that allows an arbitrary application to be executed on a range of different computational resources, providing significant flexibility for users. Scheduling decisions are often made at run time, and are based on metrics such as the availability of the computational resources, network bandwidth and any necessary data files as well as trying to match application characteristics to the most suitable platforms.

When the input files are very large, or when fault tolerance is important, users may distribute complete replicas of the data to multiple Grid nodes. Such data replication is often facilitated by special middleware such as the Storage Resource Broker from SDSC and the Replication Services from Globus. Whilst powerful, existing middleware suffers from two shortcomings. First, it typically requires modification to existing applications, which is problematic when large and complex legacy codes are executed. Many applications are too fragile to be readily adapted. Second, if the user wishes to optimize application performance, they must choose the replica manually as there is no automatic selection. This is complex and error prone, and also does not allow for

performance changes that might occur during program execution.

“Grid enabling” legacy applications is not a simple task. Grid standards such as the Web Services Resource Framework (WSRF), as exemplified by the Globus Toolkit (GT4), offer APIs that provide a fairly low level of functionality. They also require fairly substantial modifications to existing codes. Accordingly, a number of systems, such as NinfG [24], P-Grade [18], Condor-G [22], GrADS [9], APST [10], Nimrod/G [6] and GriddLeS [5], attempt to bridge the gap between high level applications and the available Grid middleware.

In GriddLeS we provide a range of mechanisms that allow a legacy application to access local and remote files, and also other applications using pipes [4][5]. These mechanisms are built on conventional file access primitives, so whilst a code might “think” it is reading a local file, it could be accessing a remote file or even data that is being written by another application on another server. The system makes it possible to build flexible Grid pipelines involving more than one computational process.

We have developed an extension to GriddLeS to allow it to access replicated data, thus addressing the two shortcomings cited above. First, GriddLeS sits above existing data replication services and provides transparent access to files using conventional file access primitives. Thus, legacy codes do not need to be modified. Second, GriddLeS monitors parameters such as the instantaneous network bandwidth and resource availability, and dynamically chooses the most appropriate replica. In this way it can choose the most appropriate data replica at run time. Moreover, GriddLeS continuously monitors these parameters and, when replica data is read only, it switches sources, allowing it to adapt to dynamic changes in bandwidth. The GriddLeS Replication Service (GRS) uses the Network Weather Service to monitor the state of the network [20].

In this paper we demonstrate the effectiveness of the GriddLeS Replication Service using two case studies. One is a simple IO bound file copy, whilst the other is

computationally bound application (a regional weather model) running on an enterprise Grid. In both cases, the input files are available on subset of the grid nodes, and GriddLeS performs the mapping between the application and the data files transparently. We demonstrate that the system is able to adapt to changes in network bandwidth during the execution and that it is able to optimize the overall performance.

2 Data Replication Services

There are several Grid data replication systems already available. Grid Datafarm [17], initiated in Japan, is an architecture designed to handle hundreds of terabytes to petabytes of data using a global distributed file system. An implementation of this architecture, called Gfarm, has been developed. It manages the data distribution in the Gfarm file system by a metadata management system and the gfarm filesystem daemon. The metadata system contains the mappings from logical file names to the physical file names. A daemon provides access control to the remote files and monitors remote resources. A parallel I/O library is available for faster data analysis. Higher-level scientific applications can be constructed through the APIs.

The Globus Replica Location Service (RLS) is an implementation of the RLS framework [1] and is available as one of the components in the open source Globus Toolkit [12][31]. RLS is a distributed replica management system designed to replace the centralized Globus Replica Catalog in the previous version of Globus. In addition, the RLS is designed to tackle issues including scalability, reliability and security. The key feature of the RLS is to maintain and provide access to mapping information from unique logical file names to the physical copies. The locations of the data copies are recorded through the distributed registry service. RLS information is stored in an open source relational database such as MySQL [27] and PostgreSQL [28].

Besides the Globus RLS, there are other tools that are implemented based on the RLS framework, such as Reptor [15] and GDMP [2] from the European Union Data Grid Project [29] as well as the File Copy and Registration Service (CAR) [30].

The Storage Resource Broker (SRB) [3] is a client-server data management system. In addition to providing mechanisms to manage replicated datasets, SRB also allows programs to access data based on the attributes rather than filenames or physical locations. SRB employs a Metadata Catalog (MCAT) service which allows users to store metadata information for the datasets. The metadata is basically information that identifies and describes the associated data copies. Information contained in the metadata may include, but is not limited

to, the physical location of the files, access control information and description of the data. SRB provides several different clients, including a set of command line tools called Scomands, a web portal called MySRB, and a Microsoft Windows client called inQ. There are also comprehensive APIs including C, C++, Java and Python.

SRB provides tools, such as srbIO and unixIO, to enable programs to access SRB data with small code modification. However, most of the replica systems require modification to an existing program that is originally not designed to support replicated data. Since code modification is error prone and is sometimes infeasible (e.g. no source code is available), there is a need for a tool that provides the existing software access to replicated data without code modification.

Further, certain levels of optimization should occur in order to minimize the program execution time. When a number of replicas are available, the application should decide which one is the best one to access and it should also switch to another one dynamically if a better one is found in run time.

In spite of these goals, very few of the replication systems provide optimization based on the resource status on the application level. Replica selection is done by the user and there is no support for dynamic replica selection during program execution. Some research aimed at providing mechanisms for replica selection [21][23][25] but currently there is little support for dynamic replica selection.

3 Introduction to GriddLeS

GriddLeS provides a general environment for the composition of Grid application from legacy code. The core concept in GriddLeS is GridFiles, a device which enables file based interprocess communication between software components. GridFile are supported by a component called the File Multiplexer (FM), as shown in Figure 1.

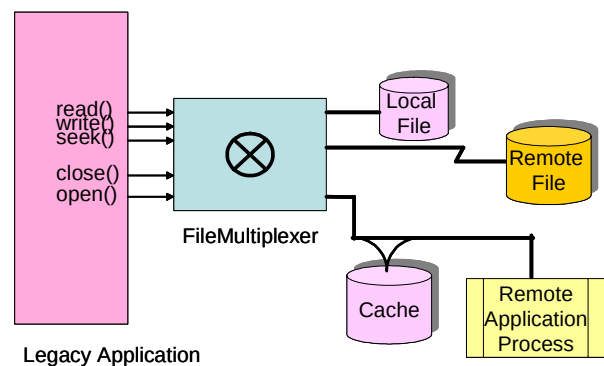
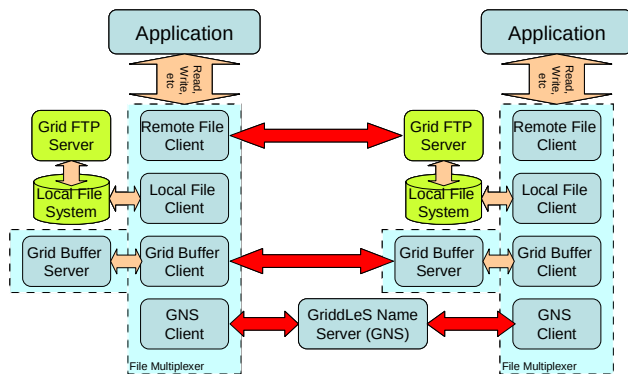


Figure 1 – The GriddLeS Architecture

When a program reads data, the FM can redirect the read operation to read from a local file, a remote 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, 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.



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. Currently, the GNS is loaded based on the structure of a Grid workflow, and this can be specified using the Kepler tool [11].

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. When the data is read, it can be sent to a cache file, and the data in the hash table will be deleted. The cache file is important because it allows a reader to reread data from previous blocks (e.g. seek). The cache file also enables broadcasting, in which multiple readers are allowed to read data written by a single writer. In this case, the first reader retrieves the data from the hash table and subsequent readers get the data from the cache file. If no cache file is used, the Grid Buffer Client only allows sequential read and write operations. 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.

4 The GriddLeS Replication Service

The GRS provides mechanisms that allow an existing program to access replicated data without code modification. It is essentially a layer that sits between the legacy application and the storage resources, as shown in Figure 3.

Users can define mapping to a SRB replica in the GNS using this format:

```
<server>:<collection>:<object ID>:<resource>
```

The *server* is the SRB server and the *collection* is the SRB collection that contains the replica. *Object ID* is the identifier of a replica that is stored in the SRB server. The *resource* is the name of the SRB resource that contains the actual replica.

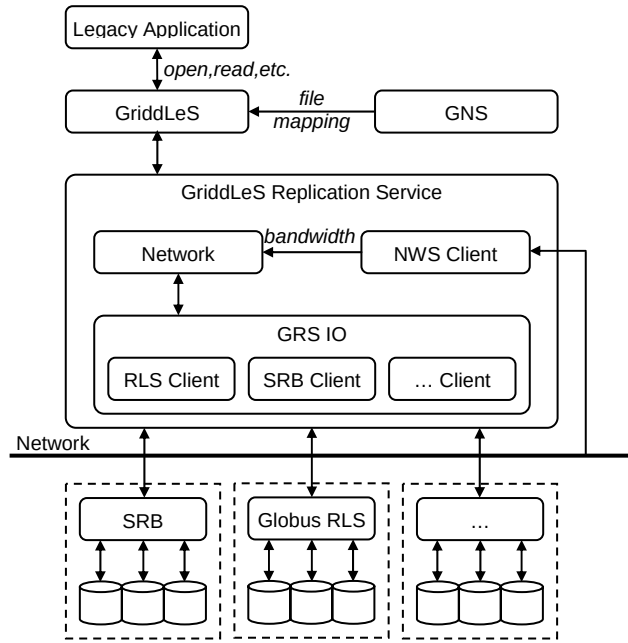


Figure 3 – Architecture of GRS

The GRS connects to a given SRB server using standard SRB login mechanisms, and thus the GNS does not require additional login information. This means that the `.MdasEnv` and `.MdasAuth` files must exist under the `.srb` directory. The former contains login information such as username, host address, host port and authorization scheme. The latter simply contains the password for login purpose.

Importantly, the GRS can choose which replica to address at run time, and this information can be hidden from the application. This provides a mechanism for optimizing the performance of the system by allowing GRS to not only choose an appropriate replica when the file is opened, but also to change that choice should network conditions change during the execution.

A NWS monitor is implemented as an important component of the GRS. This monitor is responsible for retrieving the network bandwidth information continuously when a SRB replica is opened. Bandwidth measurements between the SRB resources and the host running the program are retrieved periodically. Based on these measurements, the monitor will then forecast the bandwidth of each network and, finally, select and notify GRS the best resource for the next read call.

When an open call is intercepted, GriddleS decides which file is to be opened based on the mappings in the

GNS. If a SRB replica mapping is defined, GriddleS passes the open call to the GRS. The GRS currently connects to the given SRB server using standard SRB password authentication. After the connection is made, instead of immediately opens the file, GRS communicates to the SRB server and tries to find out if there are other replicas exist in different servers. As a result, a list of all the replicas is generated. GRS then opens the replicas and maintains a list of opened replicas. This is important because the GRS must ensure at least one replica in the list is accessible. After the replicas are opened successfully, the network monitor executes in a separate thread and monitors the network status to each replica. Note that when replica exists in the localhost, no network monitor will be run and the local replica is used. The GRS normally uses the server with the best instantaneous and predicted bandwidth, thus server changes occur depending on the current network conditions. Importantly, this change is transparent to the running application.

Since each replica is located on a different server, and a server change may occur dynamically at anytime during execution, proper synchronization is required to avoid reading incorrect blocks of data when a server change occurs. Accordingly, each time a read is performed, the new offset of the replica is calculated and stored. Upon next read, if there is no change of the resource, then GRS simply calls the usual read routines. However, when there is a resource change, GRS seeks to the correct offset before reading from the replica.

When closing, the GRS closes all the opened replicas as well as the connections to them. Also, the network monitor is killed.

Currently, the GRS only supports access to SRB datasets but we plan to support other replica systems in the near future, including the Globus RLS. The following IO functions are provided by GRS:

```
int grs_open(const char *path, int flags, mode_t mode);
ssize_t grs_read(int fd, void *buf, size_t count);
ssize_t grs_write(int fd, const void *buf, size_t count);
int grs_stat(const char *path, struct stat *buf);
int grs_fstat(int fd, struct stat *buf);
off_t grs_lseek(int fd, off_t offset, int whence);
int grs_close(int fd);
```

These functions have the same arguments and return values to the corresponding system calls. When replica functions are required, GriddleS simply forwards the arguments from the system calls to the GRS IO routines. The `grs_open` function returns a GRS-specific unique replica identifier instead of a normal system file descriptor. This identifier allows the GRS to identify the associated replicas in the subsequent IO calls.

5 Case Studies

In this section we present two case studies, one IO bound and the other compute bound, to illustrate the functionality and performance of the GRS. The first of these involves a simple program that takes a copy of the replicated data, whilst the second is a regional weather model in which the input files are replicated across a number of servers. *Importantly, neither of these applications required source changes because GriddLeS hides the details of the file replication system.*

5.1 The testbed

Table 1 shows the configuration of our testbed, consisting of 10 grid resources. All machines are running SRB 3.2.1 and NWS 2.10.1, with one SRB MCAT server running on *argos*, which stores the mappings and replica details, and non-MCAT servers on the others. The testbed spanned a number of machines within our department, but also machines in the UK, Malaysia, US and Hong Kong.

Machine	Details
argos.csse.monash.edu.au	Pentium IV 2.4GHz 480MB RAM Linux
globus.csse.monash.edu.au	Pentium II 333MHz 128MB RAM Linux
mosca.csse.monash.edu.au	SPARC SUNW, Ultra-5_10 Solaris
ocean.csse.monash.edu.au	Pentium III 500MHz 384MB RAM Linux
remus.dstc.monash.edu.au	Pentium IV 2.4GHz 512MB RAM Linux
romulus.dstc.monash.edu.au	Pentium IV 2.4GHz 512MB RAM Linux
aurora.cs.usm.my	Pentium III 1.4GHz 2048MB RAM Linux
bouscat.cs.cf.ac.uk	Pentium III 1Ghz 1536MB RAM Linux
freak.ucsd.edu	Athlon XP 1800+ 768MB RAM Linux
gideon.cs.hku.hk	Pentium IV 2.26GHz 1024MB RAM Linux

Table 1 – Testbed configuration

5.2 Methodology

For each experiment, we replicated the input files to the 9 non-MCAT servers. Each experiment was run five times on *argos* and an average time was then calculated and compared. We compare the execution time on reading from a local file, reading from a remote file at two different levels of bandwidth, as well as the improvement from the dynamic replica selection.

To evaluate the improvement from the optimizations, the bandwidth between the program running host and replica servers was lowered by transmitting large amount of data using *scp* to the servers from a separate machine outside the testbed. Since *globus* and *romulus* have the highest bandwidth, we lowered their bandwidth randomly in order to see determine the effect of the network disruption.

5.3 Replicated file copy application

This case study considers a simple file copy program that reads data from one server and copies it to another machine. Multiple copies of the input file are available and the system chooses the most appropriate one at any time based on NWS statistics and predictions for the bandwidth.

Since the data is read continuously, the network conditions will greatly affect the performance if the input data is located remotely. We used a 140MB file as the input to the file copy program and the results are shown in Table 2.

<u>Bandwidth</u>	<u>Data read from</u>	<u>Average time mm:ss</u>	<u>Performance Change</u>
~8 Mbps	romulus	03:02	100%
~0.2 Mbps	aurora	29:28	971%
Variable	Variable	03:32	116%

Table 2 – details of the file copy experiments performed

When the file copy program reads data, GriddLeS intercepted the call and passes the call to the GRS, which reads the data from the best SRB resources selected by the network monitor. Since the data is read across the network, the execution time was increased from the original 9 seconds to 3 minutes. When the bandwidth between the host and the SRB resource was reduced significantly, the execution time was further increased to more than 29 minutes.

The last experiment evaluates the improvement gained by using dynamic resource selection in the GRS. The bandwidth between the host and the SRB resources were lowered randomly. The network monitor informs the GRS

when it finds a SRB resource with better network bandwidth. During run time, the GRS switches to a better server based on the decision made by the network monitor. The result shows that the performance is nearly as good as when there is no network contention.

5.4 A regional weather model

This application concerns a computationally intensive regional weather model called DARLAM [13][14]. DARLAM. Written in Fortran, the program computes temperature and wind vectors by solving a numeric model of the atmosphere. It reads data from a 190MB input file (darlam_input.nc). This input file is replicated and stored in 9 SRB resources. A SRB MCAT server running on *argos* stores the locations of the replicas. A SRB file path is defined in the GNS to redirect the system call to read from a SRB replica instead of a local file. The results are shown in Table 3.

Exp	Model runs in	Data read from	Ave time mm:ss	Perf Change
1	argos	argos	38:39	100%
2	argos	romulus	41:47	108%
3	argos	aurora	77:32	200%
4	argos	Variable	45:02	116%

Table 3 – details of the DARLAM experiments performed

In experiment 1, DARLAM was run without GriddLeS, and the input data was stored locally. The other three experiments read replicated data from SRB resources using the GRS. In all experiments, DARLAM was run on *argos*. The IO calls made by DARLAM were intercepted and replaced by the corresponding GriddLeS IO routines.

In experiment 2, DARLAM was run on *argos* and the input data was read across the network. In this case, when DARLAM opens “darlam_input.nc”, GriddLeS passes the open call to the GRS. The GRS inspects the MCAT to obtain a list of available replicas. As expected, the execution time was slightly longer than in experiment 1 because the data was sourced across the network. However, unlike the first case study, because the application was predominantly compute bound, the time only increased by about 8%.

In experiment 3, we redirected all the IO calls to *aurora*, which has a very low bandwidth (~0.2Mbps). This time, because of the dramatic loss of network bandwidth, DARLAM was over 100% slower than in experiment 1.

In experiment 4, we replicated the input file across the 9 SRB resources and the bandwidths between *argos* and the resources were randomly lowered. As observed the

execution time in this experiment was only slightly longer than experiment 2 in which no bandwidth perturbation was performed. This is because the GRS was dynamically switching to the most effective server, as is illustrated in Figure 4. The upper part of Figure 4 shows which server was used over time and the lower part of the figure shows the bandwidth data retrieved from the NWS between *argos* and the other machines. The graph only shows 5 local machines because the other 4 external machines have very low bandwidth (<1Mbps) and thus during runtime no data was read from them. The results clearly shows that the GRS is able to change the data source dynamically.

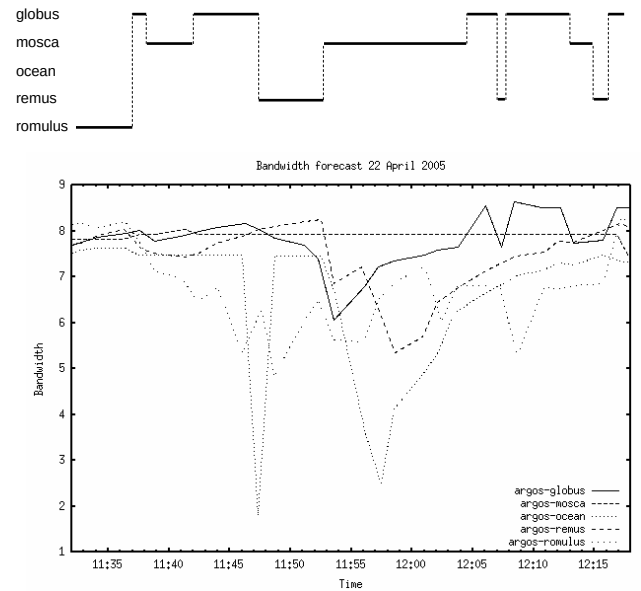


Figure 4 – Server selection against time

6 Conclusion

This paper has discussed a GriddLeS Replication Service (GRS) which allows existing applications to access replicated data. Importantly, this is performed without any modification to the application. We have described the architecture of the GRS, which provides a set of IO routines for access to different replica management systems, and have illustrated our implementation which only currently supports the SRB from SDSC. This abstraction avoids code modification because the application does not need to have specific code for any particular replica systems. In addition, the GRS provides certain level of optimization by dynamically switching to a better server. This is done by monitoring the network condition using the Network Weather Service at runtime and forecasting which connection has better bandwidth. The server selection can occur at any time during program execution to tackle unreliable network connection. This

improves fault tolerance and provides a reliable data input because the same piece of data is available in several servers. We demonstrated that by changing the data server based on the network condition can increase the overall performance.

The network monitor in the GRS currently only takes into consideration the connection bandwidth. We plan to extend it to take incorporate other resource information such as network latency. This is important, because a high latency network will lower the server response time even the bandwidth of the server is high. Load on the resource should also be considered because a high workload host may also have lower response time. Further, instantaneous network status may need to be considered because when a server suddenly goes offline, the GRS may not be able to react quickly due to the lag in the NWS client.

Moreover, we plan to support further performance optimizations such as data striping [19] and data prefetching [16]. Data striping allows parts of the dataset to be stored across the Grid for better performance. In this case, GriddLeS could provide facilities for the user to define which portions of the data are required. To further reduce the processing time, the GRS could employ data prefetching, in which the data is always read ahead and stored locally (e.g. in the memory). This can greatly eliminate the effect of high network latency. Finally, we plan to support other data replication systems such as the Globus RLS.

Acknowledgments

This project is supported by the Australian Research Council and Hewlett Packard as part of an ARC Linkage project. We wish to acknowledge our colleagues at the CSIRO, in particular John McGregor for providing access to the DARLAM model, and Jagan Kommineni who built much of the GriddLeS library. Thanks also go to the SRB team at SDSC for their assistance. Tim Ho is supported by an Australian Postgraduate Research Award (APRA-I).

References

- [1] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, B. Tierney, "Giggle: A Framework for Constructing Scalable Replica Location Services", *Proceedings of Supercomputing 2002 (SC2002)*, November 2002.
- [2] A. Samar and H. Stockinger, "Grid Data Management Pilot (GDMP): A Tool for Wide Area Replication", *IASTED International Conference on Applied Informatics (AI2001)*, Innsbruck, Austria, February 2001.
- [3] C. Baru, R. Moore, A. Rajasekar and M. Wan, "The SDSC Storage Resource Broker", *Proceeding of CASCON'98 Conference*, Toronto, Canada, 1998.
- [4] D. Abramson and J. Kommineni, "A Flexible IO Scheme for Grid Workflows", *IPDPS-04*, Santa Fe, New Mexico, April 2004.
- [5] Abramson, D., Kommineni, J., McGregor, J. and Katzfey, J. "An Atmospheric Sciences Workflow and its Implementation with Web Services", *Future Generation Computer Systems*, 21 (2005), pp 69 – 78.
- [6] D. Abramson, J. Giddy and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?", *International Parallel and Distributed Processing Symposium (IPDPS)*, pp 520- 528, Cancun, Mexico, May 2000.
- [7] D. Thain and M. Livny, "Bypass: A tool for building split execution systems", *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing*, Pittsburg, Pennsylvania, pp.79-85, August 1-4, 2000.
- [8] D. Thain and M. Livny, "Parrot: Transparent User-Level Middleware for Data-Intensive Computing", *Workshop on Adaptive Grid Middleware*, New Orleans, Louisiana, September 2003.
- [9] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski, "The GrADS Project: Software Support for High-Level Grid Application Development", *International Journal of High Performance Computing Applications*, vol. 15, no. 4, pp. 327-344, Winter 2001.
- [10] H. Casanova, G. Obertelli, F. Berman and R. Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid", *Proceedings of Super Computing 00*, Dallas, Texas, November 2000.
- [11] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher and S. Mock, "Kepler: An Extensible System for Design and Execution of Scientific Workflows", *16th International Conference on Scientific and Statistical Database Management (SSDBM'04)*, Santorini Island, Greece, 21-23 June 2004.
- [12] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", *International Journal of Supercomputer applications*, vol. 11, issue 2, pp. 115-128, 1997.
- [13] 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.
- [14] J. K. 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.
- [15] L. Guy, P. Kunszt, E. Laure, H. Stockinger, K. Stockinger, "Replica Management in Data Grids", Technical Report, Global Grid Forum - GGF5, Edinburgh, Scotland, July 2002.
- [16] M. S. Pérez, J. Carretero, F. García, J.M. Peña and V. Robles. "MAPFS: A Flexible Infrastructure for Data-Intensive Grid Applications", *Proceedings of the first European Across Grids Conference*, Febrero 2003, Spain.
- [17] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, H. Sato, Y. Tanaka, S. Sekiguchi, Y. Watase, M. Imori, T. Kobayashi, "Worldwide Grid Data Farm for Petascale Data Intensive Computing", Technical Report, Electrotechnical Laboratory, ETL-TR2001-4, 2001.
- [18] P. Kacsuk, J.C. Cunha, G. Dozsa, J. Lourenco, T. Antao and T. Fadgyas, "GRADE: A Graphical Development and Debugging Environment for Parallel Programs", *Parallel Computing Journal*, Elsevier, vol. 22, no. 13, pp. 1747-1770, February 1997.
- [19] R. Marco, J. Marco, D. Rodriguez, D. Cano, and I. Cabrillo. "RAID-1 and data stripping across the GRID", *Lecture Notes in Computer Science*, 2970:119-123, March 2004.
- [20] R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", *Journal of Future Generation Computing Systems*, Vol.15, No.5-6, pp.757-768, October, 1999.
- [21] S. Vazhkudai, S. Tuecke, I. Foster. "Replica Selection in the Globus Data Grid", *Proceedings of the first IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001)*, IEEE Computer Society Press, May 2001.
- [22] T. Tannenbaum, D. Wright, K. Miller and M. Livny, "Condor – A Distributed Job Scheduler", Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*, The MIT Press, 2002.
- [23] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. "Design of a Replica Optimisation Framework", Technical Report, DataGrid-02-TED-021215, CERN, Geneva, Switzerland, 2002.
- [24] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura and S. Matsuoka. "Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing", *Journal of Grid Computing*, Vol.1 No.1, pp.41-51, 2003.
- [25] Y. Zhao, and Y. Hu. "GRESS - a Grid Replica Selection Service", *ISCA 16th International Conference on Parallel and Distributed Computing Systems*, PDCS-2003.
- [26] Globus: GridFTP Protocol and Software, <http://www.globus.org/datagrid/gridftp.html>.
- [27] MySQL, <http://www.mysql.com>.
- [28] PostgreSQL, <http://www.postgresql.org>.
- [29] The DataGrid Project, <http://eu-datagrid.web.cern.ch/eu-datagrid/>.
- [30] The File Copy and Registration Service, <http://www.isi.edu/~shishir/car/>.
- [31] The Globus Toolkit, <http://www-unix.globus.org/toolkit/>.