HAL
open science

A Virtual File System Interface for Computational Grids

Abdulrahman Azab, Hein Meling

To cite this version:

Abdulrahman Azab, Hein Meling. A Virtual File System Interface for Computational Grids. 16th EUNICE/IFIP WG 6.6 Workshop on Networked Services and Applications - Engineering, Control and Management (EUNICE), Jun 2010, Trondheim, Norway. pp.87-96, 10.1007/978-3-642-13971-0_9 . hal-01056478

HAL Id: hal-01056478

https://inria.hal.science/hal-01056478v1

Submitted on 20 Aug 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

A Virtual File System Interface for Computational Grids

Abdulrahman Azab[1], Hein Meling [1]

[1] Dept. of Computer Science and Electrical Engineering , Faculty of Science and Technology, University of Stavanger,
4036 Stavanger, Norway
{hein.meling, abdulrahman.azab} @uis.no

Abstract. Developing applications for distributed computation is becoming increasingly popular with the advent of Grid computing. However, developing applications for the various Grid middleware environments require attaining intimate knowledge of specific development approaches, languages and frameworks. This makes it challenging for scientists and domain specialists to take advantage of Grid frameworks. In this paper, we propose a different approach for scientists to gain programmatic access to the Grid of their choice. The principle idea is to provide an abstraction layer in the form of a *virtual file system interposition agent* [1] between the local file system interface and the Grid. By abstracting away low-level Grid details, domain scientists can more easily gain access to high-performance computing resources without learning the specifics of a Grid computing platform. The proposed VFS agent is initially implemented on, HIMAN, a peer-to-peer Grid middleware. Experimental results show that the VFS only cause a negligible computational overhead to the task execution.

Keywords: Grid computing, Peer-to-peer computing, Virtualization

1 Introduction

Grid computing is an appealing concept to support the execution of large compute intensive tasks; Grid computing generally refers to the coordination of the collective processing power of scattered compute nodes to accommodate such large computations. Peer-to-peer Grid computing extends this idea, enabling all participants to provide and/or consume computational power in a decentralized fashion.

Although appealing at first sight, Grid computing frameworks are complex pieces of software and thus typically come with some obstacles limiting adoption to expert programmers capable of and willing to learn to program using the Grid middleware APIs. Therefore, a major challenge faced by developers of Grid computing frameworks is to expand the base of Grid application developers to include non-expert programmers, e.g. domain specialists using their own domain-specific tools. One

[1] The original idea of this work is the second author's

approach is to devise transparency mechanisms to hide the complexities of the Grid computing system. Similar problems are solved in other systems by interposition agents [19].

In this paper, we propose the design of a programming interface for a peer-to-peer Grid middleware through a virtual file system [1]. The proposed interface works as an *interposition agent* between the user and the client agent [3, 29]. The rationale behind this idea is that the file system interface is well-known, and is accessible from just about any domain specific tool or even simple file-based commands. The approach also lends itself to devising a file system organization exposing varying degrees of complexity to its users.

The idea is to implement the middleware core under a virtual file system layer. Thus, enabling end users/programmers to assign a task to be executed on the Grid simply by calling the `write()` function of the virtual file system on a specific virtual path. Similarly, when task execution has completed, the user can easily obtain the results by invoking the `read()` function on another virtual path. The initial implementation of the proposed VFS agent is on HIMAN [3, 29] Grid Middleware. To evaluate the performance, the execution of parallel tasks has been tested using the proposed VFS as a user interface. Results show that the VFS caused a negligible computational overhead to the task execution.

The rest of the paper is organized as follows: Section 2 gives an overview of the related efforts in the field. Section 3 presents an overview of Grid middleware architectures in general. Section 4 describes the new added virtual file system agent and how it interacts with the other components in the middleware. Section 5 presents and discusses the results obtained from the performed experiments. Section 6 presents conclusions and future work plans.

2 Related Work

The concept of virtual file systems was first introduced in the Plan 9 operating systems [1] to support a common access style to all types of IO devices, be it disks or network interfaces. This idea has since been adopted and included in the Linux kernel, and support is also available on many other platforms, including Mac OS X and Windows. There are a wide range of applications of such a virtual file system interface, including sshfs, procfs, and YouTubeFS.

2.1 Grid File systems

In the context of Grid computing, providing access to a (distributed) file system architecture is both useful and common. Transmitting the data necessary for computation is made easier with a distributed file system, and a lot of research effort has gone into data management for the Grid, e.g. BAD-FS [9], and finding efficient protocols for data transfer, e.g. GridFTP [10].

BAD-FS [9] is different from many other distributed file systems in that it gives explicit control to the application for certain policies concerning caching, consistency

and replication. A scheduler is provided with BAD-FS that takes these policies into account for different workloads.

The Globus XIO [11] provides a file system-based API for heterogeneous data transfer protocols aimed for the Grid. This allows applications to take advantage of newer (potentially) more efficient data transfer protocols in the future, assuming they too provide the XIO APIs.

These file systems tackle the problem of interacting with the data management component and file transfer protocols of a data Grid environment. However, none of them try to solve the generic interaction with the computational Grid middleware for task submission, results' collection in a suitable manner.

2.2 Interposition Agents

An interposition agent is a piece of software that inserts itself between two existing layers of software in order to modify their discourse [19]. In [17, 18], different roles of interposition agents are described and classified in more details. Some interposition agents provide integration between two applications located on the same machine. Sfio [12] is a C++ library for managing I/O streams which provides functionality similar to that of Stdio [13]. It creates an interposition layer between the user code and I/O streams. The aim is to offer more facilities and control over I/O operations together with hiding the complexity from the user. HIA [14], Heap Interposition Agent, library is a part of MemorySpace debugger [15]. HIA is inserted between the user's application code and the C malloc() API [16], and defines functions for each of the memory allocation API functions. The aim is to avoid memory allocation errors resulted from passing invalid values to the operations in the C Heap Manager API.

Other interposition agents provide seamless integration between two applications located on different machines. Parrot [18] provides seamless integration between standard Unix applications and remote storage systems. This integration makes a remote storage system appear as a file system to a Unix application. One drawback is that the user has to specify the location of the remote machine. Besides, all commands has to be included in a Parrot command, which decreases the provided transparency. Bypass [20] is a general purpose tool for building interposition agents based on split execution. Instead of locating an instance of the interposition agent on the home machine, the user interacts with a shadow process which communicates with the interposition agent on the remote machine. Eternal [23] is a CORBA [22] complaint system for providing fault-tolerance in CORBA based distributed systems. Eternal provides fault-tolerance by replicating and distributing the application objects across the system. This is carried out without modifying the ORB [21], and the programmers write the application as if it were a sequential program to be run on a single machine. Other examples are: GCB [24], DCache [25], Paradyn[26], and SOCKS[27].

All of these systems provide an intermediate layer between two applications to provide or enhance the integration or to perform an additional intermediate role. The user has to know how to deal with the front end application which is system specific. The proposed VFS interface agent provides an interface which is well known to all computer users and locates itself as the front end application. The user doesn't have to

provide any information about the remote machine. In addition, it provides a data management interface (i.e. file-system) for a computation management application (i.e. computational grid). To our knowledge, most of the existing computational Grid middleware environments [30, 31, 32, 34, 35, 36, and 37] require a special language level API for interaction with the scheduler and to collect the results.

3 Grid Middleware

The proposed VFS agent is initially implemented on HIMAN [2, 3, 4, 5, 29], a peer-to-peer Grid Middleware for accomplishing serial and parallel computational tasks. HIMAN provides three main agents for a Grid system: *Worker*, which is responsible for task execution, *Client*, which is responsible for task submission and execution monitoring, and *Broker*, which is responsible for the task allocation. There are two versions of HIMAN: The pure peer-to-peer version [2, 3, 4, 5], and the hybrid peer-to-peer version [29]. In the pure peer-to-peer version, all nodes have the three agents (i.e. worker, client, and broker) installed. The roles of the client and the broker are carried out by the submitting node which is responsible for task submission, allocation, and execution monitoring. In addition, any node can be a submitter for a task and executer for another task at the same time. In the hybrid peer-to-peer version, the Grid system is divided into a set of virtual organizations [28]. Each virtual organization contains a set of peers and one broker. All nodes have the worker and the client agents installed, but the broker will be only installed on stable and powerful nodes. Brokers from different virtual organizations construct a cooperative Broker Overlay [29]. The interaction between different components in the system for the pure, and the hybrid peer-to-peer versions are described in [3] and [29] respectively.

4 Virtualizing Grid Access

In the current version of HIMAN, as well as other existing Grid solutions, the user has to be aware of many configuration parameters and has sufficient background on distributed computing in order to execute his codes on the Grid. This can be suitable for expert users, but for regular users, a simple and familiar interface is needed. One of the most well known interfaces to all computer users is the File System interface. The proposed solution is a *Virtual File System* (VFS) agent to be placed as an interposition agent between the user FS interface and the client agent. The role of this new agent is to provide an easy to use interface between the user and the client agent, to accomplish both task submission and result reporting and collection processes.

4.1 VFS Agent

The VFS Agent is initially implemented on HIMAN as an additional interface layer above the client, and built using Callback file system [38] which is a product of ELDOS [39]. Callback is similar to FUSE (Linux user mode file system) [7], and

designed to enable running file system code in user space in Windows. A similar Windows solution is Dokan file system [6], which is less stable than callback. The Callback library contains a user mode API to communicate with user applications, and a kernel mode file system driver for communicating with the windows kernel. The role of the Callback FS in our VFS agent is to enable the front end users of Grid systems to provide inputs and collect results as files using simple FS commands. In order to make the proposed VFS applicable and easy to implement in other Grid systems which are based on different platforms, the communication between the VFS agent and the client agent is carried out through exchanging simple UDP text messages. In order to implement the VFS on any other Grid system, a small UDP communication module is needed to be added to the client agent.

The VFS agent is composed of two main modules: 1) Task Submission Module (TSM), and 2) Result Collection Module (RCM).

Task Submission Module (TSM). TSM contains two routines. The first is the virtual volume routine, and will be called once the Callback file system is mounted, and creates a virtual volume, which will be seen by the Windows explorer. This new volume will be used by the user to provide the input files. The second is the task submission routine and responsible for forwarding the input to the client agent in order to start the task submission process. We have included the task submission routine in the `CbFsFsWrite()` function in the CallBack user mode library, so that it will be triggered when a specific `write()` command is executed on the virtual drive. The default case is when the code [3, 5] file is copied to the virtual drive. This can be changed by the user to determine which `write()` command will trigger the TSM. The task submission procedure, depicted in Fig. 1, goes in the following steps:

1. The user executes a `write()` file system command, through a FS interface (e.g. Windows Explorer), to write the input files into the virtual volume.
2. The FS interface forwards the command to the Windows kernel.
3. The kernel forwards the command to the Callback file system driver.
4. The file system driver responds by calling the `CbFsFsWrite ()` function in the CallBack user mode library.
5. This function invokes the task submission routine in the TSM.
6. The task submission routine responds by presenting the Input files in the file list of the virtual volume, so that they can be accessed with the FS interface, and forwarding the inputs to the client agent.
7. The client agent submits the task to the Grid.
8. During the execution, the user can monitor the execution process (e.g. the progress) simply by executing a FS `read()` command to a text file in the virtual drive.

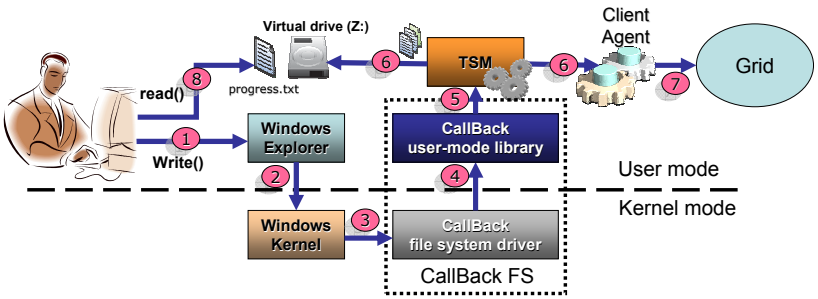


Fig. 1. Task submission procedure

Result Collection Module (RCM). The aim of the result collection procedure is to notify the user of the end of the execution and to provide a VFS interface for collecting of the task result files. The result collection procedure, depicted in Fig. 2, goes in the following steps:

1. The task execution is completed, and the results are sent to the client agent in the submitting node [5].
2. The client agent sends the collected results to the RCM.
3. The RCM responds by creating a new virtual directory (e.g. \Results) in the virtual drive, and displaying the results as virtual files in this directory. The creation of the results' directory will notify the user with the end of the task execution.
4. The user can collect the result by executing a FS `move()` command to the files in the results' directory, through the FS interface, to move the result files to any physical path.
5. Windows Explorer forwards the command to the kernel.
6. The kernel forwards the command to the CallBack file system driver.
7. The file system driver responds by calling the `CbFsRenameOrMoveEvent()` event procedure in the CallBack user mode library.
8. The `CbFsRenameOrMoveEvent()` procedure responds by moving the result files to the new physical path, and removes their specifications from the file list collection of the result collection virtual drive.

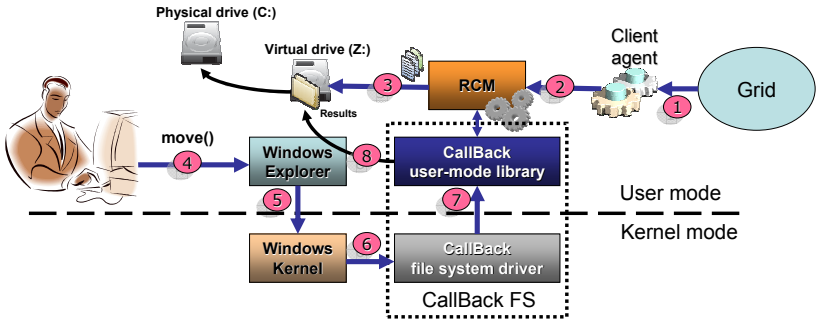


Fig. 2. Result collection procedure

4.2 A Simple Example

In this example, a practical description is given to the role of the proposed VFS in executing a parallel task. The use case is implemented on HIMAN middleware. For implementing a simple parallel task on HIMAN, the user provides the input as one code file and a number of data files equals to the number of the parallel subtasks. Assuming that the code file is <code.dll> and five input data files will be provided, <input1.data>, <input2.data>…, <input5.data>, and the user will use the DOS command prompt to issue file system commands. Assuming that the virtual drive is (Z), and the user initially stores the input files in C:\Input\. The following steps describe the whole execution procedure:

1. The user writes the data files to the virtual drive by typing:
   `> copy c:\input\*.data z:`
2. The user writes the code file to the virtual drive by typing:
   `> copy c:\input\code.dll z:`
   This command will trigger the TSM to start the task execution by the client, and create the progress virtual file <progress.txt> on Z.
3. During the execution, the user can monitor the execution progress for the subtasks by typing:
   `> type progress.txt`
   The output:
   ```
   Subtask 1.     Worker:193.227.50.201 Progress: 10%
   Subtask 2.     Worker:74.225.70.20   Progress: 15%
   Subtask 3.     Worker:87.27.40.100   Progress: 20%
   Subtask 4.     Worker:80.50.96.119   Progress: 6%
   Subtask 5.     Worker:211.54.88.200  Progress: 12%
   ```
4. The user can continue retyping the command to catch up with the execution progress. The user has nothing to do with the management issues (e.g. scheduling, fault tolerance, connectivity…etc) which are carried out by the Grid middleware [2, 3, 4, 5]. In case of a worker failure, the worker address will be changed for the associated subtask in the progress file.
5. When the execution of one or more subtasks is completed, this will be shown in the progress virtual file:

```
Subtask 1.     COMPLETED
Subtask 2.     Worker:74.225.70.20   Progress: 90%
Subtask 3.     Worker:87.27.40.100   Progress: 88%
Subtask 4.     COMPLETED
Subtask 5.     COMPLETED
```

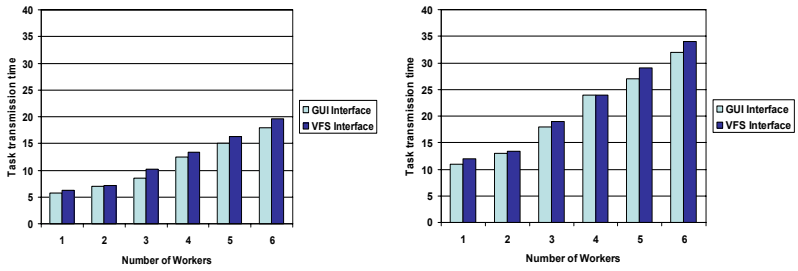
6. Upon the completion of all subtasks, the <Results> virtual directory will be created and the user can move all the virtual files in that directory to a physical directory (e.g. c:\Results\) by typing:

   > `move z:\Results\*.* c:\Results`

5 Initial Performance Evaluation

In order to prove the usability of the proposed VFS agent, it is important to show that it causes a very little overhead to Grid tasks. Since that the VFS agent is initially implemented only in HIMAN framework, the aim of the performed experiment has been to compare the transmission time of the task files in case of using the HIMAN built in client GUI to that case when the VFS interface agent is used.

Two experiments of running matrix multiplication parallel tasks have been tested for both cases for multiplying two square matrices with a size of: a) 1500x1500 and, b) 2100x2100. For both experiments, different number of workers (i.e. parallel subtasks) varying from one to six has been used. The results are depicted in Fig. 3.



a) Matrix size 1500x1500 b) Matrix size 2100x2100

Fig. 3. Task transmission time against the number of workers using the client GUI and the VFS interface

In Fig. 3, it is clear that the computational overhead which is represented by the increase in the task transmission time in case of using the VFS interface agent instead of the built in client GUI, is nearly negligible. Since the transmission of the input files to workers is done in a serial fashion, the overhead is due to the time taken by the VFS to build the virtual files in the memory and to communicate with the client agent for transmitting each of input files.

6 Conclusions and Future Work

According to the complexity of Grid systems, providing a useful and easy to use interface to computational Grid is a big challenge. This paper has proposed a new technique which provides the usage of a computational Grid middleware through virtual file system interface. The idea is to include the proposed interface as an added component to existing computational Grid middleware's. By applying this technique to the HIMAN middleware, it has been described how the new component works as an interface between non-expert Grid users and the middleware core components. Initial evaluation results show that the computational overhead which is resulted from using the VFS interface agent instead of the built in client GUI, is negligible.

It is planned to provide the same interface with few changes for parallel task execution in HIMAN. The proposed technique can be applied to other computational Grid solutions, such as Globus, Condor, and Alchemi. The proposed interface can also be implemented using FUSE for use in Linux based Grid solutions.

References

1. R. Pike, D. Presotto, K. Thompson, H. Trickey, and P. Winterbottom. The use of name spaces in Plan 9. SIGOPS Oper. Syst. Rev., 27(2):72 -76, 1993.
2. Kholidy, H.A. Azab, A.A. Deif, S.H. Enhanced "ULTRA GRIDSEC": Enhancing High Performance Symmetric Key Cryptography Schema Using Pure Peer To Peer Computational Grid Middleware (HIMAN). ICPCA 2008. Vol. 1, page(s): 26 – 31, 2008.
3. El-Desoky, A.E. Ali, H.A. Azab, A.A. A Pure Peer-To-Peer Desktop Grid framework with efficient fault tolerance. ICCES'07. page(s): 346 – 352, 2007.
4. Azab, A.A. Kholidy, H.A. An adaptive decentralized scheduling mechanism for peer-to-peer Desktop Grids. ICCES'08. page(s): 364 – 371, 2008.
5. El-Desoky, A.E. Ali, H.A. Azab, A.A. Improving Fault Tolerance in Desktop Grids Based On Incremental Checkpointing. ICCES'06. page(s): 386 – 392, 2006.
6. Dokan. http://dokan-dev.net/en/. Retrieved: 20 June 2009.
7. Filesystem in Userspace. http://fuse.sourceforge.net/. Retrieved: 20 June 2009.
8. .Net Framework 2.0 Solution Center. http://support.microsoft.com/ph/8291. 20 June 2009.
9. J. Bent, D. Thain, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny. Explicit Control in a Batch-Aware Distributed File System. In Proceedings of the First USENIX Symposium on Networked Systems Design and Implementation (NSDI '04), San Francisco, CA, March 2004.
10. GridFTP. http://globus.org/toolkit/data/gridftp/. 20 June 2009.
11. W. Allcock, J. Bresnahan, R. Kettimuthu, and J. Link. The globus extensible input/output system (xio): A protocol independent io system for the grid. In IPDPS '05: Proceedings of

the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 4, page 179.1, Washington, DC, USA, 2005. IEEE Computer Society.

12. Kiem-Phong Vo, ``The Discipline and Method Architecture for Reusable Libraries", Software - Practice & Experience, v.30, pp.107-128, 2000.

13. Glenn S. Fowler, David G. Korn, Kiem-Phong Vo. SFIO: A Safe/Fast I/O Library. http://www.research.att.com/~gsf/download/ref/sfio/sfio.html.

14. Chris Gottbrath. Memory Debugging In Parallel and Distributed Applications. 2nd Parallel Tools Workshop HLRS. Stuttgart, Germany. July 8th 2008.

15. MemoryScape. http://www.totalviewtech.com/products/memoryscape.html.

16. malloc: Allocate Memory Block. http://www.cplusplus.com/reference/clibrary/cstdlib/malloc/.

17. Sander Klous, Jaime Frey, Se-Chang Son, Douglas Thain, Alain Roy, Miron Livny and Jo van den Brand. Transparent access to Grid resources for user software. *Concurrency Computat.: Pract. Exper.* 2006; 18:787–801.

18. Douglas Thain and Miron Livny. Parrot: Transparent User-Level Middleware for Data-Intensive Computing. Scalable Computing: Practice and Experience, Volume 6, Number 3, Pages 9-18, 2005.

19. Michael B. Jones. Interposition Agents: Transparently Interposing User Code at the System Interface. fourteenth ACM symposium on Operating systems principles. Asheville, North Carolina, United States. Pages: 80 – 93, 1994.

20. Douglas Thain and Miron Livny, "Multiple Bypass: Interposition Agents for Distributed Computing", *The Journal of* Cluster Computing, Volume 4, 2001, pp 39-47.

21. Object Management Group. The Common Object Request Broker: Architecture and Specification, Revision 2.0, 1995.

22. J. Siegel, CORBA Fundamentals and Programming, John Wiley & Sons Publishers, New York, 1996.

23. P. Narasimhan, L. E. Moser, P. M. Melliar-Smith. Exploiting the Internet Inter-ORB Protocol Interface to Provide CORBA with Fault Tolerance.Usenix Conference on Object-Oriented Technologies and Systems. Portland, Oregon, June 1997.

24. Son S, Livny M. Recovering internet symmetry in distributed computing. Proceedings of CCGrid. IEEE Computer Society Press: Los Alamitos, CA, 2003.

25. M. Ernst, P. Fuhrmann, M. Gasthuber, T. Mkrtchyan, and C. Waldman. dCache, a distributed storage data caching system. In Proceedings of Computing in High Energy Physics, Beijing, China, 2001.

26. B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. B. Irvin, K. Karavanic, K. Kunchithapadam, and T. Newhall. The Paradyn parallel performance measurement tools. IEEE Computer, 28(11):37–46, November 1995.

27. M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS protocol version 5. Internet Engineering Task Force (IETF) Request for Commends (RFC) 1928,March 1996.

28. I. Foster, C. Kesselman, S. Tuecke.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In: International J. Supercomputer Applications, 15(3), 2001.

29. Abdulrahman Azab, Hein Meling. Decentralized Service Allocation in a Broker Overlay Based Grid. In Proc CloudCom 2009, Beijing, China, 1-4 Dec., 2009, pp. 200-211.

30. Condor project. http://www.cs.wisc.edu/condor/.

31. The Globus toolkit. http://www.globus.org/toolkit/.

32. Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal. Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework. Wiley Press, New Jersey, USA, June 2005.

33. EGEE: Enabling Grids for E-Science in Europe. http://public.eu-egee.org/.

34. D4Science: Distributed collaboratories Infrastructure on Grid ENabled Technology 4 Science. http://www.d4science.eu/.

35. Gardner R. Grid3 : An Application Grid Laboratory for Science. In Proc. Computing in High Energy Physics and Nuclear Physics 2004, Interlaken, Switzerland, 27 Sep - 1 Oct, 2004, pp.18.

36. Hermann Lederer, Gavin J. Pringle, Denis Girou, Marc-Andre Hermanns, Giovanni Erbacci. DEISA: Extreme Computing in an Advanced Supercomputing Environment. NIC Series, 2007, 38(1): 687-688.

37. NorduGrid: Nordic Testbed for Wide Area Computing and Data Handling, http://www.nordugrid.org/.

38. Callback File System. http://www.eldos.com/cbfs/.

39. ELDOS CORP. Solutions For File and Document Storage Development. http://www.eldos.com/.