Middleware of Taiwan UniGrid

Po-Chi Shih[1], Hsi-Min Chen[2], Yeh-Ching Chung[1], Chien-Min Wang[3], Ruay-Shiung Chang[4], Ching-Hsien Hsu[5], Kuo-Chan Huang[6], Chao-Tung Yang[7]

shedoh@sslab.cs.nthu.edu.tw, seeme@selab.csie.ncu.edu.tw, ychung@cs.nthu.edu.tw, cmwang@iis.sinica.edu.tw, rschang@mail.ndhu.edu.tw, chh@chu.edu.tw, kchuang@mail.ntcu.edu.tw, ctyang@thu.edu.tw

[1]Department of Computer Science
National Tsing Hua University

[2]Department of Computer Science and Information Engineering
National Central University

[3]Institute of Information Science
Academia Sinica

[4]Department of Computer Science and Information Engineering
National Dong Hwa University

[5]Department of Computer Science and Information Engineering
Chung Hua University

[6]Department of Computer and Information Science
National Taichung University

[7]Department of Computer Science and Information Engineering
Tunghai University

ABSTRACT

Taiwan UniGrid (Taiwan University Grid) is a Grid computing platform, which is founded by a community of educational and research organizations interested in Grid computing technologies in Taiwan. In this paper, we present the design and development of a middleware for Taiwan UniGrid. Taiwan UniGrid middleware consists of three primary modules: 1) UniGrid Portal, 2) Computing Service, and 3) Data Service. We explain the major design issues that we suffered from the development of these three modules and propose the corresponding approaches to them. The detailed system architecture, software components and features are elaborated. Finally, an example of a workflow consisting of MPI parallel jobs demonstrates that users can utilize Grid resources with ease via our middleware platform.

Keywords

Grid middleware, portal, computing grid, data grid.

1. INTRODUCTION

With the rapid growth of computing power, storage capacity and network speed, many researchers and scientists have been concentrated on the development of various Grid systems to efficiently utilize distributed computing and storage resources for large scale applications in the last decade. In Taiwan, a community of educational and research organizations interested in Grid computing technologies founded a Grid computing platform, called Taiwan UniGrid (abbreviation of Taiwan University Grid) [1]. These organizations devote their computers/clusters to Taiwan UniGrid for sharing and collaboration with each other. The objective of Taiwan UniGrid is to provide education and research organizations with a powerful computing platform where they can study Grid-related issues, practice parallel programming on Grid environments and execute computing/data-intensive applications. So far, over than 30 institutes participate in Taiwan UniGrid and contribute their resources with approximately 80 hosts and 180 CPUs.

Taiwan UniGrid is constituted of three primary modules: 1) UniGrid Portal, 2) Computing Service, and 3) Data Service. In the following, we will present the major issues that we suffered from the development of these three modules and propose the corresponding approaches to them.

UniGrid Portal is a Web interface that bridges users and underlying services and resources. It serves as a Grid desktop providing users with working spaces to submit jobs, manage archived data, and check statuses of resources and submitted jobs. The first development consideration of Portal is the issue of Single Sign-On. Without Single Sign-On, users have to keep a list of accounts for each machine by themselves. This becomes an obstacle for users to use Grid systems. The other consideration of Portal development is workflows of jobs. Since large-scale applications may consist of a number of structured jobs executed in sequential or in parallel, it is inconvenient for users to submit jobs one by one manually.

Computing Service deals with job execution processes from submissions to obtaining results. It organizes distributed and heterogeneous computing resources and performs job scheduling to select suitable resources for job requests. For various service purposes, Computing Service integrates a set of existing production software. In order to organize scattered computing resources as a unified computing platform and interact with other Grid systems, we adopt Globus Toolkit [3] as our underlying computing middleware. We use Ganglia [4] and NWS (Network Weather Service) [5] to provide detailed static and dynamic resource information (ex. number of CPUs, CPU loading, provided and available memory) and network states for job scheduling. We also install Condor-G [6] on each site for local fault-tolerance. Although Globus Toolkit provides GRAM (Grid Resource Allocation Management) service to facilitate job

SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.

submissions, it does not support job scheduling in global level and leaves it to the development of upper-layer service. To solve this problem, we propose a *global scheduler* to discover resources and allocate proper ones for submitted job

Data Service [8] is built on top of SRB (Storage Resource Broker) [9], which is a general data management tool integrating distributed and heterogeneous storage resources and providing virtualized access interface. SRB has been a production data management tool and adopted by several Grid projects. Thus, we decide to build our Data Service based on SRB while developing additional features that are not well supported by SRB. These features includes 1) *high-performance data transfer* which speed up the data transfer rate in data replication, downloading, moving, and copying; 2) *data sharing* which helps users share their data in a manner of forming user groups and specifying admission control on each data object; 3) *graphic data management client* by which users can manage their data transparently as using local file systems.

The remainder of the paper is organized as follows. In Section 2, we explain the literature related to Grid middleware. Section 3 describes the middleware architecture and its features in detail. A workflow example is demonstrated in Section 4. Finally, we present some concluding remarks in the last section.

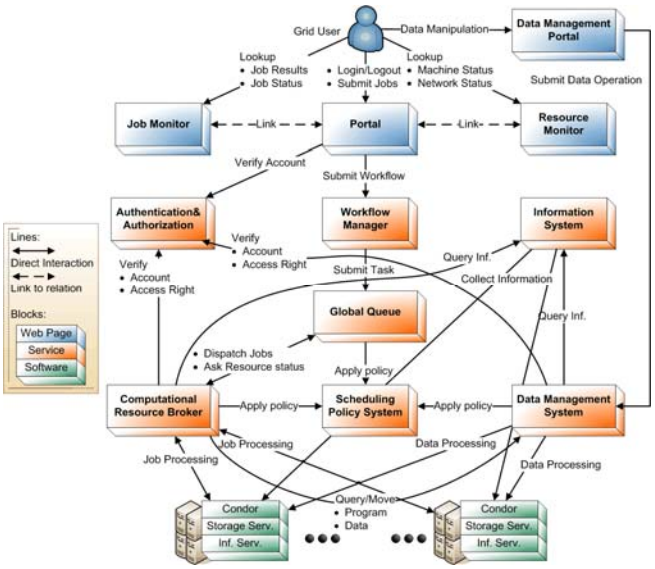

Figure 1. Component diagram for Taiwan UniGrid.

2. Related Work

InteGrade [10] is an object-oriented Grid middleware that leverages the idle computing power of shared commodity machines to perform parallel applications. It adopts CORBA technology to enable communications between distributed service components via CORBA IDL interfaces. Taiwan UniGrid middleware is based on standard Web Service technology [11, 12, 13] and WSRF [14] that define standard conventions to implement services and resources so that they can be easily integrated. On the other hand, since InteGrade targeted at a lightweight Grid middleware, the issues of user authentication and data management, such as data replication and file movement, are not well addressed.

MyGrid [15] is a service-based Grid middleware to support *in silico* experiments in biology. The advanced feature of MyGrid is that it uses semantic web technology to manage annotation, ontologies and semantic discovery. The services of workflow management and distributed database query are provided by MyGrid to form biological experiments. The major difference between MyGrid and Taiwan UniGrid is that the workflow supported by MyGrid is composed of services wrapping bioinformatics tools, while that supported by Taiwan UniGrid is made up of general jobs.

GridLab [16] is another service-based Grid middleware funded by EU. GridLab divides system hierarchy into two spaces: user space in high layer and capability space in low layer. Capability space consists of services and resources, while Grid Application Toolkit (GAT) API is resided in user space to manage and invoke underlying resources and services for Grid applications. However, data management supported by GridLab is limited in comparison with SRB adopted by Taiwan UniGrid.

3. SYSTEM ARCHITECTURE

Figure 1 shows the detailed component diagram of Taiwan UniGrid middleware. The blue blocks provide Web interfaces for users to utilize underlying resources. The orange blocks are software services that manage Grid resources and cooperate with each other to provide computing and data manipulation functionalities. Green blocks list the software installed on each participated machine. Beside these components, we exploit MySQL database [17] to store all required information accessed by services. The functionality of each component is described as follows:

- Portal – a Web interface to submit job (form as workflow) to *workflow manager*.
- Data Management Portal – a Java Web-Start client interface to upload/download data to/from underlying storage system.
- Job Monitor – a Web interface to check out the historical job statuses or cancel jobs in execution.
- Resource Monitor – a Web interface to display detailed text and graphical resource information.
- Authentication & Authorization – provides Single Sign-On mechanism to validate users.
- Workflow Manager (WM) – handles job dependency and flow control, and submits jobs to *Global Queue*.
- Information System (IS) – collects resource information accessed by other components
- Global Queue (GQ) – provides a centralized queue for all independent jobs received from *Workflow Manager* and schedules them by using *Scheduling Policy System*.
- Scheduling Policy System (SPS) – performs job scheduling and resource allocation algorithms on *global queue* and dispatches scheduled jobs to *Computational Resource Broker*.
- Computational Resource Broker (CRB) – processes job allocation to scheduled *condor* server and feedback real-time job execution statuses to *Job Monitor*.
- Data Management System (DMS) – performs data management operations.
- Condor – local job scheduler installed in each individual site controlling which local resources are allocated to execute jobs.
- Storage Service – SRB server.
- Information Service – Ganglia gmond service.

3.1 UniGrid Portal

In essence, services and resources within a Grid system are distributed in different organizations and managed independently. Each organization has its own security policy. Without Single Sign-On, users have to keep a list of accounts for each machine by themselves. This becomes an obstacle for users to use Grid systems. Therefore, we propose a Single Sign-On mechanism, as shown in Figure 2, based on GSI [2] to generate proxy certificate for each individual user and store it in Globus MyProxy server. By doing so, users only need to login once in Portal and utilize services and resources without concerning the authentication settings of them. Computational Resource Broker and Data Management System will take care of certificate mapping and validations for users.

The second consideration of Portal development is workflows of jobs. Since large-scale applications may consist of a number of structured jobs executed in sequential or in parallel, it is inconvenient for users to submit jobs one by one manually. Inspiring from this requirement, we build a workflow management tool in our middleware. Through the tool, users can edit job requests to form a workflow in Portal and submit the workflow to be executed automatically by back-end computing services. The workflow structure is constructed by stages and jobs where each stage contains independent jobs and the relations between stages are dependent (the jobs in next stage must be submitted only when all jobs in current stage are finished). While users press "submit" bottom from Portal, an XML-formatted workflow is sent to Workflow Manager which parses the workflow and controls the job execution.

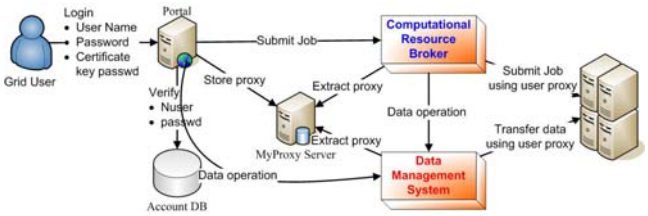

Figure 2. Single sign-on mechanism.

3.2 Computing Service

Computing Service is responsible for all job-related operations through the collaboration of three components, i.e. Global Queue, Computational Resource Broker, and Scheduling Policy System. Workflow Manager stores all independent jobs into Database. Then, Global Queue provide a service, called JobDBCrawler, will periodically query Database for new incoming jobs and put them in a job queue. While any new job queried, SPSController service will be notified by JobDBCrawler to request Scheduling Policy System for scheduling. Besides, SPSController can also be notified when any pre-executed job finished since allocated resources can be released for another resource match opportunity.

The design of isolating Workflow Manager and Computing Service via Database provides failure recover capability for underlying computing services. For instance, if Global Queue, Computational Resource Broker, or Scheduling Policy System is failed, jobs submitted by users can still store in DB with job status marked as waiting. As long as computing services recover on-line, Global Queue resume crawling jobs from Database, and then all pre-scheduled and executed but unfinished jobs will be rescheduled by Computational Resource Broker and Scheduling Policy System.

Scheduling Policy System is designed to handle scheduling and allocation processes. Scheduling controls the job order in queue and allocation decides which resource to be allocated. The required resource information is queried from Information Service to filter out resources that does not satisfy job requirements. This component plays a major role on system performance so that the design of the algorithm was another major research topic which is discussed in [7]. In short, we implement FCFS and narrowest job first (the job request least CPUs) scheduling algorithm and best-fit/fastest single (allow parallel execution on single site)/cross (allow for cross-site parallel execution) site algorithm for system administrator to choose proper configuration.

After scheduling, the scheduled jobs will be dispatched to Computational Resource Broker for proceeding job dispatch and execution. Computational Resource Broker is implemented with multiple threads that each thread is responsible for serving one single job until it finished. The job execution flow includes 1) obtain user proxy certificate, 2) transfer required files, 3) execute job, and 4) retrieve result. All screen outputs will be treated as job results and all output files specified in job requirements will be saved into Data Service after job finish.

Figure 3 shows the sequence diagram describing the interactions between components of Computing Service. The red line indicates false notification that could be ignored. Besides, parameters related to environment and system are defined in a configuration file by which developers can migrate middleware to different platform (ex. service ip, port, or file location) with flexibility and adjust system behavior more adaptable (ex. number of job queue or running thread, failure retry times, or proxy life time).

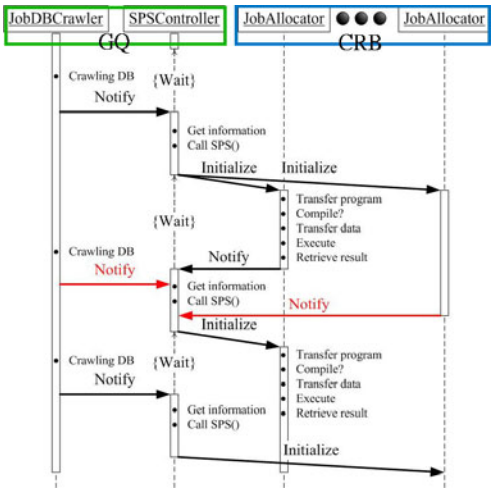

Figure 3. Sequence diagram of computing service.

3.3 Data Service

Figure 4 shows the framework of Data Service. In the server side, the left bottom of the framework is a set of physical storage resources, including hard disks, tapes and databases, contributed by the members of Taiwan UniGrid. We adopt SRB as a data management middleware to integrate these scattered storage resources. Although SRB has furnished an efficient data transfer

approach by using multiple TCP streaming, we propose an alternative, called *Self Adaptation* [18], to achieve a higher data transfer rate in comparison with the original one. We also add the alternative (Self Adaptation patch) into the original functions of SRB. A set of extended SRB APIs are built on top of SRB and the *Self Adaptation* patch..

The right of the server side of the framework is a number of Web services used for data management. We adopt Web services technologies [9, 10, 11] in our Data Service to integrate other software developed by third parties. There are two services implemented in the current system: *AutoReplicator* service and *Account Management* service. Grid users can utilize *AutoReplicator* service to set various replication policies. *Account Management* service wraps up the functions of user authentication in UniGrid Portal for enabling Single Sign-On.

In the client side, the bottom is the data management library for Taiwan UniGrid which connects to the corresponding server-side extended SRB APIs and data management services. We implemented two versions of the library. One is Java-based and another is C-based. The data management library provides a uniform interface of data and storage management by which programmers can build various Grid applications to access the underling storage resources.

According to the literature survey, we found that Grid users usually need an environment where they can work collaboratively. Although most Data Grid middleware provides the sharing of storage resources, data sharing for collaborative work is not well supported. Therefore, in our system, we develop a collaborative mechanism in our Data service through the combinations of forming user groups and specifying access permissions on each data object.

Figure 5 presents the deployment of our Data Service. Since there is a huge amount of storage resources distributed in Taiwan UniGrid, using a single information server to maintain the metadata regarding users, data and storages may cause the problems of server overloading and single point of failure. To avoid these problems, we divided all storage resources in Taiwan UniGrid into five zones. Each zone has a MCAT (SRB Metadata Catalog) server installed for maintaining the metadata. To enable the flexibility of sharing, the administrators of a MCAT server can specify their own sharing policies. In addition, each MCAT server periodically synchronizes its metadata with each other to keep the metadata consistency among zones. By synchronization, Grid users registered in one zone can access any storage resources located in other zones and retrieve sharing data timely.

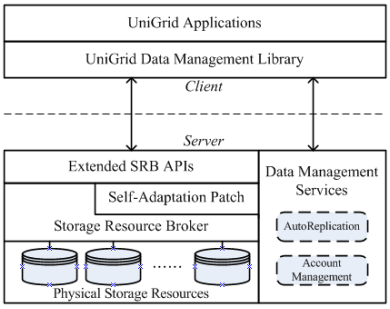

Figure 4. The framework of the virtual storage system for Taiwan UniGrid.

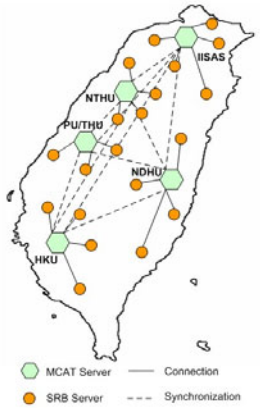

Figure 5. The deployment of the virtual storage system for Taiwan UniGrid.

4. DEMOSTRATION

In this section, we use a workflow example, which composes of two stages as depicted in Figure 6, to demonstrate our middleware. An MPI parallel job is executed in stage 1, and a sequential job written in C is run in stage 2. The rectangle box represents the executable jobs and ellipse box represent the data file.

Figure 7 shows the workflow management interface and detailed job description in stage 1. The scheduler option indicates that a job is submitted to local condor scheduler rather than GRAM. Sort Key is used for SPS to determine which resource is preferable for this job. There are three options, CPU bound, memory bound, and network bound to support decision making. Run Type determines which kind of application the job is since different instructions are required to perform job execution. SRB Directory indicates if the file is stored in SRB server. If checked, CRB will transfer the specified file from user home directory of SRB to the scheduled resources before job execution. Similarly, if output filename is specified, CRB will save the specified file from the scheduled site back to user home directory of SRB after the job is executed successfully. Machine Select is an option for users to manually select resources without SPS.

In order to demonstrate cross site parallel execution, we manually select two physically distributed sites *iisgrid* (from Academia Sinica in Taipei) with 4 CPUs and *zeta1* (from THU in Tainchung) with 2 CPUS. This *hellompi_out1* MPI program will save the hostname on each executed resource to *data1.out* file. The job in stage 2 *inout* merges the content of two files specified in argument and generate *result.out* file.

After building up the workflow, there are three actions that users have to perform:, 1)upload the input file (via the data management client provided by Data Service as shown in Figure 9), 2)submit the workflow (simply click submit button in Portal),and 3)wait for results. Figure 8 shows the monitoring page of job status which lists all historical job information submitted by users including job name, job start and end time, current status, and result. The result is saved as file which could be seen in Figure 9.

This demonstration is an example that illustrates how easy users can utilize the large amount of Grid resources; meanwhile, provide users with flexible configuration to submit various kinds of applications.

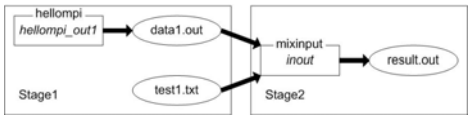

Figure 6. Demo workflow.

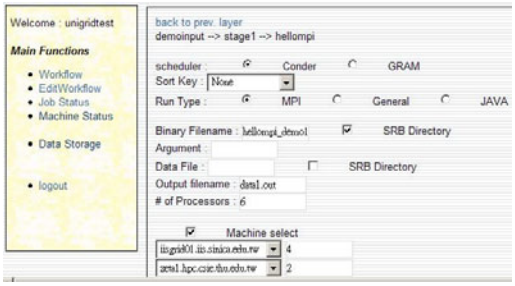

Figure 7. Job description in stage 1.

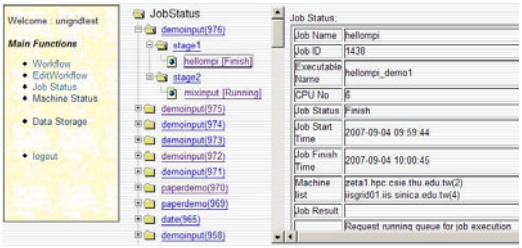

Figure 8. Job status monitoring pages

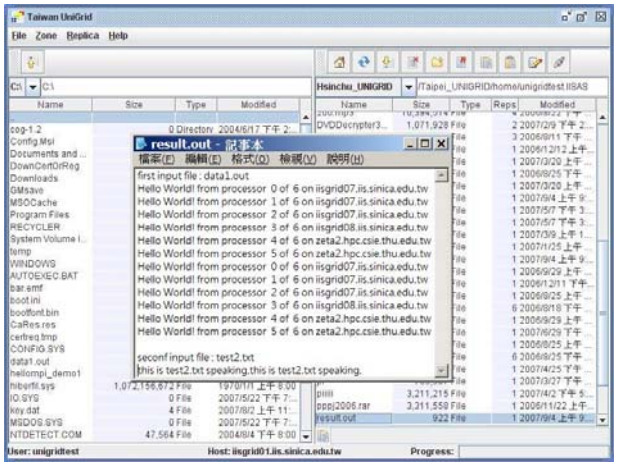

Figure 9. A screenshot of the data management client.

5. CONCLUSION

In this paper, we present the design and development of a middleware for Taiwan UniGrid. We explain the major design issues that we suffered from the development of the middleware. Single Sign-On and workflow management are two major considerations we face in the development of UniGrid Portal. We take the issues of integration of software and resources, and global scheduling into account when we build Computing Service. In Data Service, we concern with high performance data transfer and data sharing. The detailed system architecture, software components and features are elaborated. Finally, we use an application example to demonstrate our middleware by which users can access the resources in Taiwan UniGrid with ease. In future work, we supposed to integrate peer-to-peer technology in the design of grid middleware to enhance the fault-tolerance capability and negate the disadvantages of centralize approach.

6. REFERENCES

[1] Taiwan UniGrid, http://www.unigrid.org.tw

[2] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," *In ACM Conference on Computers and Security*, ACM Press, 1998.

[3] Ian Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, No. 2, 1997.

[4] Ganglia, http://www.ganglia.info/

[5] Network Weather Service (NWS), http://nws.cs.ucsb.edu/

[6] Condor-G, http://www.cs.wisc.edu/condor/condorg/

[7] Kuo-Chan Huang, P.C. Shih, and Y.C. Chung, "Towards Feasible and Effective Load Sharing in a Heterogeneous Computational Grid,", *Second International Conference on Grid and Pervasive Computing*, 2007

[8] Chien-Min Wang, H.M. Chen, C.C. Hsu, J.J. Wu, "A High-Performance Virtual Storage System for Taiwan UniGrid", *Second International Conference on Grid and Pervasive Computing*, 2007

[9] Chaitanya Baru, R. Moore, A. Rajasekar and M. Wan, "The SDSC Storage Resource Broker," *CASCON '98: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research*, Canada, 1998, also available at http://www.sdsc.edu/srb.

[10] Andrei Goldchlegery, F Kon, A. Goldman, M. Finger, and G. C. Bezerra, "InteGrade: Object-oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines," *Concurrency and Computation: Practice & Experience*, Vol. 16, March, 2004. John Wiley and Sons.

[11] WSDL: Web Services Description Language 1.1. Available at http://www.w3.org/TR/wsdl

[12] UDDI: Universal Description, Discovery and Integration. Available at http://www.uddi.org, 2001.

[13] SOAP: Simple Object Access Protocol 1.1. Global Grid Forum, available at http://www.w3.org/TR/soap

[14] Globus Alliance, IBM, and HP. Web Service Resource Framework. http://www.globus.org/wsrf. 2004.

[15] L. B. Costa, L. Feitosa, E. Araujo, G. Mendes, R. Coelho, W. Cirne, and D. Fireman, "MyGrid: A Complete Solution for Running Bag-of-tasks Applications," *In Proc. of 22nd Brazilian Symposium on Computer Networks – III Special Tools Session*, May 2004.

[16] Allen, G., K. Davis, K. N. Dolkas, N. D. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, and I. Taylor, "Enabling Applications on the Grid - A GridLab Overview," *International Journal of High Performance Computing Applications*, 11, vol. 1, 2003..

[17] MySQL: http:// www.mysql.com/

[18] Chien-Min Wang, C.C. Hsu, H.M. Chen, J.J. Wu, "Efficient Multi-source Data Transfer in Data Grids," *6th IEEE International Symposium on Cluster Computing and the Grid*, Singapore, May 2006