

GRESS – a Grid Replica Selection Service

Yong Zhao

Department of Computer Science
University of Chicago
yongzh@cs.uchicago.edu

Yu Hu

Department of Computer Science
University of Chicago
yuhu@cs.uchicago.edu

Abstract

Grid technologies and infrastructures facilitate distributed resource sharing and coordination in dynamic, heterogeneous, multi-institutional environments. A replica catalog is a Grid component that keeps replica locations of data objects and provides location transparency to data access. Replica selection is of great importance to data-intensive scientific computing targeted by many data Grid projects and the enabling virtual data technologies. We present here GRESS, a Grid replica selection service that provides a consistent interface to multiple replica selection implementations and supports flexible result delivery and easy configuration and deployment of new replica selection algorithms. GRESS is based on the Open Grid Services Architecture, which facilitates discovery and incorporation of the service by other Grid components such as Grid planners and virtual data workflow execution environments. We discuss the motivation for and implementation of GRESS and describe its role in a Grid environment.

Keywords: Grid Computing, Replica Selection, OGSA, Web Service, Virtual Data, GRESS.

1. Introduction

Grid computing, the integration of a collection of distributed computing resources to offer performance unattainable by any single machine, has experienced a surprisingly fast evolution in many fields during the past several years. The Globus Toolkit® [1] enables the construction of computational Grids by providing a set of service components such as authentication, communication, information service, and data access.

Besides these components, however, higher-level services are required for wide-area practical computing problems. In particular, a replica management service is critical. Such a service [2] comprises two components: a replica location service (RLS) and a replica selection service (RSS).

One promising step toward developing an effective RLS is the system Giggle [3]. Giggle is a framework with which one can tune the behavior of the RLS system based on the scale, performance, reliability, and cost requirements of particular classes of application.

The development of a practical RSS, on the other hand, remains a problem. Replica selection involves choosing a replica from among those spread across the Grid, based on some application-specified characteristics [4]. Many approaches have been proposed to address the replica selection problem [5] [6]. Although these approaches differ in implementation and performance, they share a common framework: data collecting, preprocessing, and predicting. Their disadvantage rests with the fact that they all need to set up a Grid environment and deploy their replica selection modules, a process that is tedious.

In this paper, we present a new Grid replica selection service, called GRESS, that minimizes this effort and unifies different replica selection approaches into a generic platform. GRESS supports easy incorporation of various replica selection implementations and provides a consistent interface to them. The system is built as a Grid service based on the Open Grid Services Architecture (OGSA) [7], which facilitates discovery and incorporation of the service by other Grid components.

The rest of the paper is organized as follows. In Section 2, we discuss the importance of replica selection and different replica selection algorithms. In Section 3, we introduce OGSA and Grid services and explain the motivation in building GRESS as a Grid service. In Section 4, we describe the architecture and implementation of GRESS. In Section 5, we summarize our work on GRESS and briefly describe future research and development.

2. Replica Selection

Replica selection is essential to data-intensive scientific applications that are of primary concern by a number of data Grid projects, such as the Grid Physics Network [8], the Particle Physics Data Grid (www.ppdg.net/), and the

EU DataGrid project (www.eu-datagrid.org/).

Virtual data technologies [2] provide data location transparency and materialization transparency in which replica service plays a central role. Upon request for a data product, the system decides whether to transfer the data to the user or to reproduce the data, depending on which is faster. In the case of reproduction, the system can choose to move data close to a computation site or to move the computation close to the data, depending on the cost. In either case, the cost estimation is based on which replica is to be selected. Replica selection is also important in Grid planning and scheduling [9].

The key to replica selection is the prediction of file transfer time, which depends on different factors including characteristics of transfer, network status, server load, and disk I/O information. Shen and Choudhary [10] construct performance models of system components for making performance predictions; however, it is difficult in practice to get accurate predictions from these models [11]. Other tools such as the Network Weather Service (NWS) [12] base their predictions on historic information at end-to-end level. Although NWS achieves good results in predicting network bandwidth, however, it appears to be unsatisfactory when applied to GridFTP data transfers [6]. Combining data from GridFTP log file, NWS, and disk I/O information, Vazhkudai and Schopf have formulated a regression predictor [6] that is more accurate but needs much history data and processing.

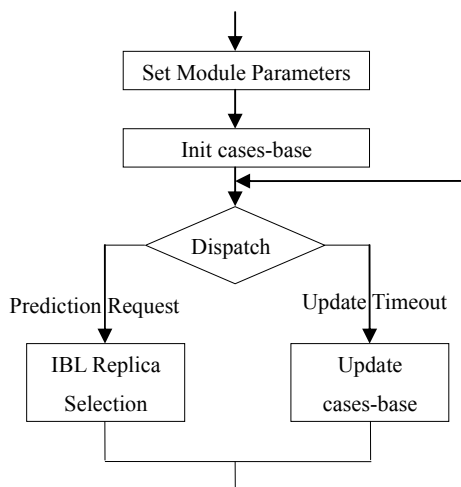


Figure 1: IBL Replica Selection Module

We use a light-weighted algorithm proposed by Hu and Schopf [5] as the default replica selection module in GRESS. It adopts instance-based learning (IBL)

technology to make selections based only on transfer log files. The algorithm achieves a reasonable performance and has the added advantages of simpler cases-base and a lighter-weight selection process.

The IBL replica selection module comprises four components as shown in Figure 1: parameter setting, initialization of cases-base, IBL replica selection, and update of cases-base. During the parameter setting process, values such as weights and number of near neighbors are assigned. The initialization component constructs the cases-base by storing a number of historic transfer log files (training instances). Now the module can predict which server is best, whenever an attempted file transfer request (requesting instance) comes. The selection is based on the similarity between the requesting instance and training instances. The cases-base is updated periodically in order to keep its freshness.

3. OGSA and Grid Service

OGSA represents a natural evolution of Grid technologies, in particular the Globus Toolkit, merged with concepts and technologies from the Web services communities. OGSA adopts a common representation for computational and storage resources, networks, programs, databases, and the like. All are treated as services: network-enabled entities that provide some capability through the exchange of messages [13].

OGSA defines uniform service semantics (the Grid service) and standard mechanisms for creating, naming, and discovering transient Grid service instances. It also provides location transparency and multiple protocol bindings for service instances, and supports integration with underlying native platform facilities [13].

Grid services provide (in terms of the Web Service Description Language) a set of well-defined interfaces and follow specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability. The conventions address naming and upgradability. Grid services also address authentication, authorization, delegation and concurrency control [14]. Figure 2 shows the structure of a Grid service.

We base GRESS on OGSA and build it as a Grid service for the following reasons:

- It allows easy discovery through community registries. Service providers can publish detailed descriptions about the service.
- It expresses service functions in a consistent

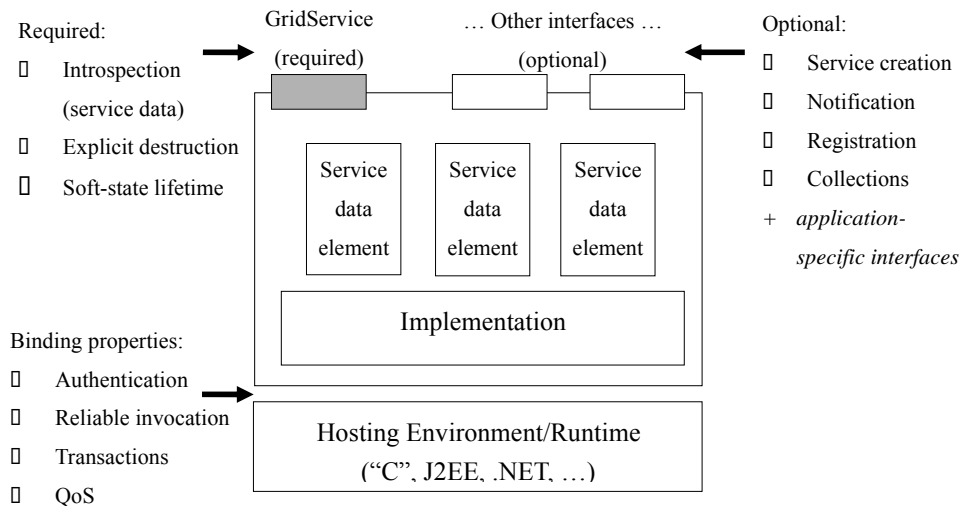


Figure 2: Grid Service Structure

form, and supports diverse selection modules with different performance characteristics.

- It supports different transport protocol bindings such as HTTP and SOAP...
- It provides flexible result delivery.
- It supports easy configuration and deployment of new selection modules so that they can be plugged in dynamically.
- It can be easily incorporated into or composed with other Grid components, such as Grid planners or virtual data workflow execution environments.

4. Architecture and Implementation

We show in Figure 3 the architecture of GRESS and how it interacts with the user application. The primary function GRESS provides is to select a replica location given a logical file name (LFN). To achieve this, it incorporates at least one replica selection module to choose the “best” replica. We can define *best* from different perspectives, for example, the shortest transfer time or the most stable server. Accordingly, we want to have different replica selection modules that focus on different performance characteristics. Therefore another function of GRESS is to configure and deploy alternative modules in the service.

4.1 Use of GRESS

A user application (a program, a service, etc.) firstly contacts the community registry to find a replica selection service provider, the registry returns the handle of the master replica selection (MRS) factory service, which is the entry point of GRESS, to the user.

The user then sends a request to the MRS factory to create an MRS instance and gets back the Grid service handle (GSH) and Grid service reference (GSR) for the new service instance. A GSH is a globally unique name that distinguishes a specific Grid service instance from all other Grid service instances. The GSH carries no protocol- or instance-specific information such as network address and supported protocol bindings. Instead, this information is encapsulated, along with all other instance-specific information required to interact with a specific service instance, into a GSR. Unlike a GSH, which is invariant, the GSR(s) for a Grid service instance can change over that service’s lifetime [7].

If the user wants to incorporate a module into GRESS, the user calls the *ConfigureModule* operation provided by the MRS instance. This operation makes necessary changes to a template RS service implementation, and then generates a new RS service, including the corresponding RS factory for the module. The *DeployModule* operation is then called to deploy the newly generated service and the module in GRESS.

Now GRESS is ready to help the user make replica selection decisions. The simplest case is to find the best replica location for a file, given the logical file name. Since in practice different virtual organizations maintain their own RLS, a RLS server contact (typically an URL) is also required.

The MRS instance queries the RLS for the replicas of the logical file, which returns a list of physical file names (PFNs). The MRS instance then makes a request to the RS factory to create a new RS instance and forwards the list of PFNs to the RS instance, which in turn calls the

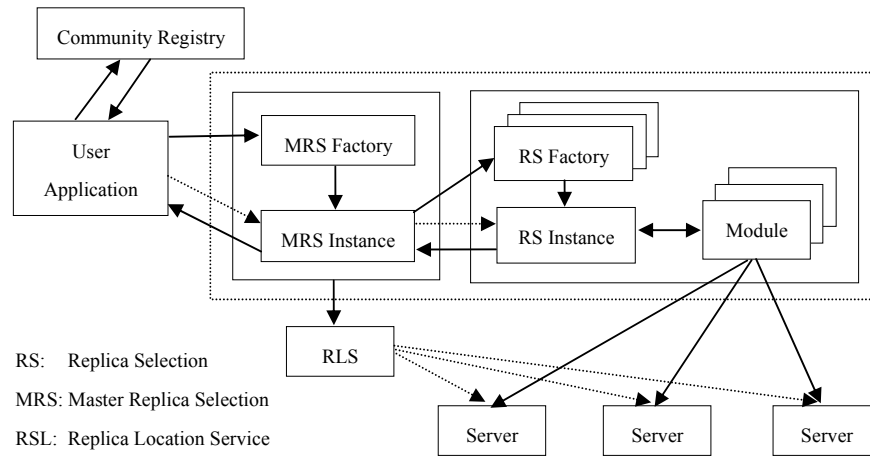


Figure 3: GRESS Architecture

module to make the selection. The module estimates the time to transfer the file from different replica locations and selects the fastest one. The result, one of the PFNs, is returned to the RS instance, then to the MRS instance, and finally to the user.

GRESS is designed to handle more than the simplest case. One of the key features of GRESS is to facilitate easy configuration and deployment of new replica selection modules and allow the user to experiment with the replica selection algorithm without wasting time setting up the experiment platform. When multiple modules are deployed in the system, the user can explicitly choose which module to use. Alternatively, the user can specify a set of categories, such as nonintrusive, short response time, or short transfer time, and the system chooses the right module that satisfies these categories.

GRESS also has flexible result delivery mechanisms. The user application can get immediate results by calling operations defined in the service interface. It can also query the service data elements (SDE) of the MRS instance using standard operation *FindServiceData* defined in the obligatory GridService interface. SDE is a set of named and typed XML elements encapsulated in a standard container format; it provides standard representation for information about Grid service instances. This operation provides a pull model for delivery of service data.

Moreover, OGSA provides a notification framework that allows clients to register interest in being notified of particular messages and supports asynchronous, one-way delivery of such notifications. This is the push model for delivery of service data. It is useful when the client wants to be notified of important changes.

4.2 Organization of GRESS

This section focuses on the organization of the four layers in GRESS. To make information exchange simple, we define all the correspondence to be a string of the format:

Attribute_Name = Attribute_Value, ..., Attribute_Name = Attribute_Value;

Attribute name-value pairs are separated by a comma, and the string ends with a semicolon. There can be white spaces and line separators before the semicolon. A sample message looks like this:

ServerName = gargoye.cs.uchicago.edu, FileSize = 32, TransferTime = 15;

MRS is the component that directly interacts with the user application. It allows the user to configure, deploy, select and use a module. The functions in a module are mapped to standard service interfaces. The replica selection result is delivered by MRS to the user.

The interface between MRS and RS is straightforward. MRS maintains a list of active RS instances and forwards user requests to the right RS instance. RS is responsible for interacting with the module, that is, calling the mapped functions according to the user's request and gathering the results from the module.

Usually the module requires some initialization parameters to get started (for example, IBL needs four weights assigned to day, time, mode and file size). Thus, we have the *SetModuleParameters* operation in RS. The parameters are usually stored in a parameter file.

Results from the module are passed back to RS in special strings mentioned earlier. Getting back result from synchronous calls is easy. On the other hand, we can envision modules that operate asynchronously: that is, they start the test and return immediately, and it takes

time to get the result. In this case, we provide a callback mechanism. The module must call the *CollectResult* shell script (which is preset to invoke the callback function provided by RS) and send back the result.

4.3 Implementation of GRESS

We implement GRESS using the Globus Toolkit version 3 (GT3), a full open source OGSA implementation. GT3 is composed of GT3 core, which implements the Grid service interfaces and behaviors; GT3 base services, which exploit the GT3 core to implement both existing Globus Toolkit capabilities (e.g., resource management and data transfer) and new capabilities (e.g., reservation and monitoring); and higher-level services (such as job manager or reliable file transfer).

GRESS is written in Java; the IBL module is written in Perl. GT3 provides development toolkits to generate the WSDL definition, as well as the server-side and client-side stub implementations for the Grid service from a Java interface. The service implementation and configuration is packed into a Grid Archive Package and deployed in a Grid service hosting environment. In our case, we used both the GT3 standalone service container and Apache Tomcat servlet container.

We also provide a Web interface that allows the user to make requests to GRESS by filling out simple forms. shows a screenshot of the Web page.

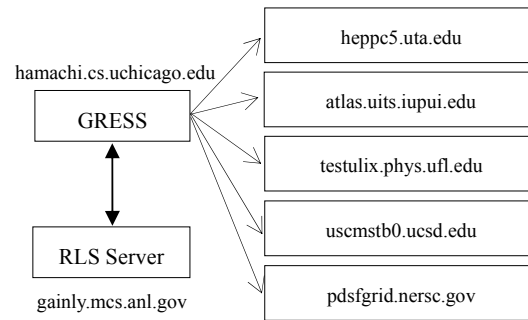


Figure 5: GRESS Experiment Environment

We gathered some preliminary results. For example, in Figure 6 we show the estimated and actual transfer times for a 32 MB file transfer from the servers, where server1 to server5 correspond to the five servers in Figure 5 top down.

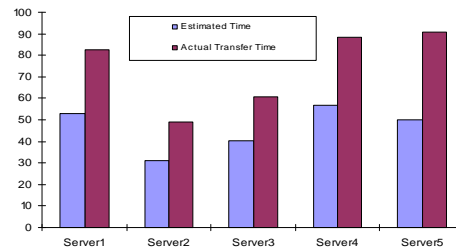


Figure 6: Transfer Time Estimation

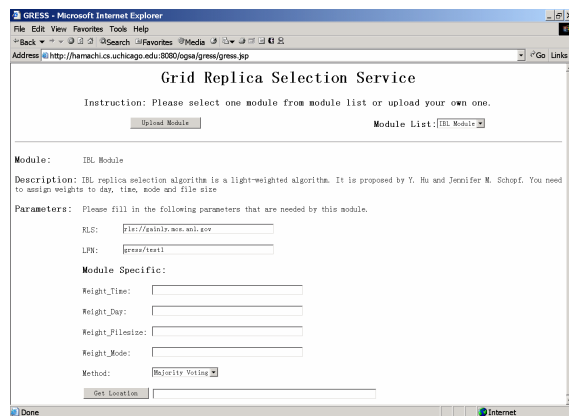


Figure 4: Screenshot of the GRESS Web Page

We set up an experiment environment as shown in Figure 5, where GT3 and GRESS were installed at *hamachi.cs.uchicago.edu*. The RLS server was set up at *gainly.mcs.anl.gov*, and we registered a number of files with various sizes and replica locations to the RLS server.

Similar results can be obtained for other deployed modules, from which we can analyze the differences in performance and characteristics of these modules. We also are integrating more replica selection modules into the system. Our intention is to compare their behaviors and performances at varying server loads and network conditions.

5. Summary and Future Work

We have presented GRESS, an OGSA-based Grid replica selection service. Our design allows for the easy plug-in of new replica selection modules with minimum modification and facilitates integration with other Grid components. Modules can be configured to share useful log data, and applications can choose the modules that best suit their demands. We have implemented the system using the GT3 OGSA implementation and tested it in a Grid environment.

We expect to further enhance GRESS and incorporate it into data Grid applications. In particular, we are

interested in the effect of caching the results returned by modules. Caching could significantly improve the response time, especially for modules that take longer time to make selection choice. It may not be preferable, however, when the network or server conditions change dynamically. Also, the refresh rate of the cache needs to be fine-tuned.

Another interesting issue concerns choosing the right module for different applications. Our module categorization mechanisms are still not sufficiently flexible to match application demands and module features. We envision a profile-based matching mechanism, where our service keeps statistical information about each module and build a profile for each module.

We plan to enrich the result set and provide visualization interface, so that the performance of modules can be better represented and understood, which can help test and improve the modules.

Acknowledgment

We thank Ian Foster, Gail Pieper and Jennifer Schopf for their comments and suggestions. This research was in part supported by the National Science Foundation's GriPhyN project under contract ITR-0086044.

References

- [1] I. Foster, C. Kesselman. The Globus Project: A Status Report. *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, pp. 4-18, 1998.
- [2] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 23:187-200, 2001
- [3] A. Chervenak, E. Deelman, I. Foster, et al. Giggie: A Framework for Constructing Scalable Replica Location Services. *Proceedings of Supercomputing 2002 (SC2002)*, November 2002.
- [4] 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)*, pp. 106-113, IEEE Computer Society Press, May 2001.
- [5] Y. Hu, IBL for replica selection in data intensive Grid applications, Master's Thesis, Department of Computer Science, University of Chicago, 2003.
- [6] S. Vazhkudai, J. Schopf, Using Regression Techniques to Predict Large Data Transfers, *Submitted to the Journal of High Performance Computing Applications - Special Issue on Grid Computing: Infrastructure and Applications*.
- [7] I. Foster, C. Kesselman, J. Nick, S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum*, June 22, 2002.
- [8] P. Avery and I. Foster, The GriPhyN Project: Towards Petascale Virtual Data Grids, 2001. www.griphyn.org.
- [9] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A Virtual Data System for Representing, Querying and Automating Data Derivation. *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, Edinburgh, Scotland, July 2002.
- [10] X. Shen and A. Choudhary. A Multi-Storage Resource Architecture and I/O, Performance Prediction for Scientific Computing. *9th IEEE Symposium on High Performance Distributed Computing*. 2000: IEEE Press.
- [11] J. Geisler and V. Taylor. Performance Coupling: Case Studies for Measuring the Interactions of Kernels in Modern Applications. *SPEC Workshop on Performance Evaluation with Realistic Applications*, 1999.
- [12] R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. *Cluster Computing*, 1998.
- [13] I. Foster, C. Kesselman, J. Nick, S. Tuecke. Grid Services for Distributed System Integration. *Computer*, 35(6), 2002.
- [14] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, Grid Service Specification. Open Grid Service Infrastructure WG, Global Grid Forum, Draft 2, 7/17/2002.
- [15] S. Graham et al., Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, Sams Technical Publishing, Indianapolis, Ind., 2001.
- [16] E. Christensen et al., Web Services Description Language (WSDL) 1.1, W3C Note, Mar. 2001; <http://www.w3.org/TR/wsdl> (current June 2003).