GridSolve: The Evolution of A Network Enabled Solver

Asim YarKhan, Jack Dongarra, and Keith Seymour

Innovative Computing Laboratory,
Department of Computer Science,
University of Tennessee, Knoxville, TN, USA
{yarkhan,dongarra,seymour}@cs.utk.edu

Abstract. GridSolve is a stubless RPC-based client-agent-server system for remotely accessing hardware and software resources. GridSolve emphasizes ease-of-use for the user and includes resource monitoring, scheduling and service-level fault-tolerance. In addition to providing Fortran and C clients, GridSolve enables SCEs (such as Matlab) to be used as clients, so domain scientists can use Grid resources from within their preferred environments. GridSolve is a more highly evolved version of the earlier NetSolve project, and it is based on the emerging GridRPC standard. This paper will discuss the changes and improvements involved in the evolution from NetSolve to GridSolve.

1 Introduction: The Grid and Network Enabled Solvers

The adoption of Grid infrastructures as a major platform for supercomputing holds great promise for accelerating scientific discovery. However, the use of Grid infrastructures has, for the most part, been restricted to the largest and most resource intensive projects. For Grid computing to become a true success story, it must become an infrastructure that can be easily used by the *general* community of scientist and engineers. Within this community of practitioners, the use of scientific computing environments (SCEs) such as Matlab or Mathematica is pervasive. These domain specialists are accustomed to the flexible computing environment provided by an SCE, which gives them with the tools and libraries that they need to be productive and enables them to go from computation to visualization in an natural fashion.

Network enabled solvers can be used to extend the power of SCEs so that they reach beyond the users desk, and into the network of resources available on the Grid. End users are not required to install and maintain local software and libraries, and can simply use the libraries that have been installed at a remote location. Since the libraries and remote services can be maintained by experts, they can be highly tuned and provide the optimized execution on the remote platform.

The purpose of GridSolve is to create the middleware necessary to provide a seamless bridge between the simple, standard programming interfaces and

desktop systems that dominate the work of computational scientists and the rich supply of services supported by the emerging Grid architecture, so that the users of the former can easily access and reap the benefits (shared processing, storage, software, data resources, etc.) of using the latter. The vision of the broad community of scientists, engineers, research professionals and students, working with the powerful and flexible tool set provided by their familiar scientific computing environments, and yet able to easily draw on the vast, shared resources of the Grid for unique or exceptional resource needs, or to collaborate intensively with colleagues in other organizations and locations, is the vision that GridSolve is designed to realize.

2 Foundations of GridSolve: GridRPC and NetSolve

GridSolve is based on the RPC paradigm for distributed computing, but it is an entire environment which provides stubless clients, resource discovery, load balancing, fault tolerance, asynchronous calls, disconnected operation and security. A primary goal for GridSolve is ease-of-use, providing transparent access to resources. GridSolve employs two primary enabling technologies, the NetSolve solver [2] and the GridRPC API [11].

2.1 GridRPC: An API for Grid Remote Procedure Calls

The GridRPC API represents ongoing work to standardize and implement a portable and simple remote procedure call (RPC) mechanism for Grid computing. This standardization effort is being pursued through the Grid Remote Procedure Call Working Group within the Open Grid Forum (formerly Global Grid Forum). GridRPC provides a common setting within which users can develop RPC programs, so that these programs are source code compatible. GridSolve has recently passed a GridRPC compliance test, along with two other GridRPC implementations, Ninf-G [12] and DIET [4].

2.2 NetSolve: A Precursor to GridSolve

NetSolve is a client-agent-server system which provides remote access to hardware and software resources through a variety of client interfaces. A NetSolve system consists of three entities, as illustrated in Figure 1.

- The **Client**, which needs to execute some remote procedure call. NetSolve client interfaces have been implemented in Matlab, Mathematica, Octave, C, Fortran and Java. Client-side stubs are not required to access remote services, the client-side service bindings are looked up from the server as needed.
- The **Server** executes services on behalf of the clients. The server hardware can range in complexity from a uniprocessor to a MPP system and the functions executed by the server can be arbitrarily complex. Server administrators

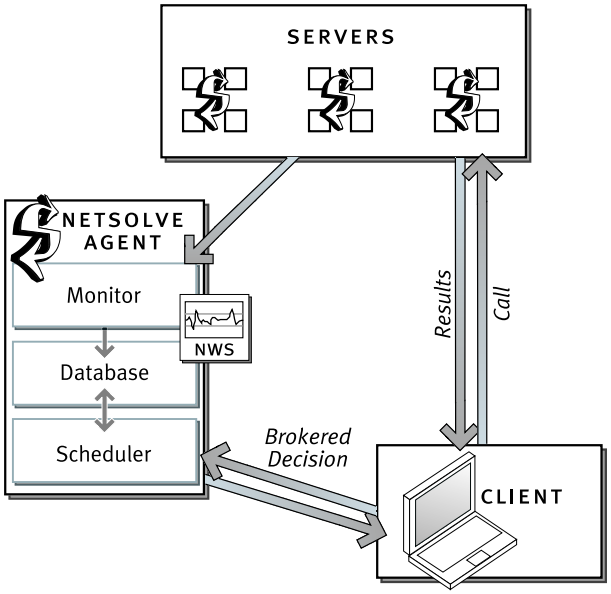

Fig. 1. NetSolve client-agent-server architecture. The agent *monitors* the servers on the Grid and records available service and server status in a *database*. The agent can also record network status using NWS (Network Weather Service). The agent *schedules* the client request to be executed on an appropriate server. The GridSolve system uses the same architectural model.

can write service definitions and add their own services without affecting the rest of the NetSolve system. Since there are no client side stubs, any client can become immediately aware of any services that are added.

– The **Agent** is the focal point of the NetSolve system. It maintains a list of all available servers and services, and performs resource selection and scheduling for client requests as well as ensuring load balancing of the servers.

The system is designed to be easy to use from the perspective of an end-user. The user executes code containing a call to NetSolve similar to `netsl('myfunction',parameters, ..)`. The rest of the remote execution happens transparently from the point of view of the client. The GridSolve client library contacts the agent which finds servers that can satisfy the request, and ranks these servers. The client receives the list of server and submits the request to highest ranked server. If the request fails for some reason (e.g. network problems, server down), the client can automatically resubmit to the next server in the list, providing service level fault tolerance. The server executes the requested service and returns the output to the client. In this way, the end user can access Grid resources without having to be aware of all the details involved

in finding, allocating and monitoring the resources and managing the software and libraries.

In addition to providing the middleware necessary to perform the brokered remote procedure call, NetSolve provides mechanisms to interface with other existing Grid services. NetSolve can use server-proxies to communicate with several back-end resource and execution managers, such as batch queue managers, the Condor [9] high throughput computing system, and MPI runtime systems. A server-proxy is specific to a back-end system, and accepts requests from the client using the same protocols as a standard NetSolve server. The primary benefit is that the client-to-server communication protocol is identical so the client does not need to be aware of every possible back-end service. The actual resources that execute a service may be a serial machine or a parallel machine, with the same service being implemented using different algorithms on different servers.

NetSolve has several specialized execution mechanisms which support common computing models. There is a task-farming API within NetSolve that supports parameter-sweep or master-worker style applications. A task sequencing API enables workflow type applications where the input data or intermediate outputs are to be retained at the remote server, and to be used in further computation.

NetSolve is distributed with service wrappers for many numerical libraries, such as LAPACK, ScaLAPACK, SuperLU, ARPACK and PETSc. If these libraries are available at the servers, they can be enabled within NetSolve. Some numerical libraries (e.g., BLAS, LAPACK, SuperLU) are even included in the NetSolve distribution, enabling a NetSolve server to provide useful services immediately upon installation.

2.3 Shortcomings of NetSolve

Network design, hardware architectures, and software methodologies have changed substantially since the beginning of the NetSolve project in 1996 [5]. More and more sites are using NATs (Network Address Translators) as a method of extending IP usage within a private subnet and as a security tool. NetSolve was designed before the widespread use of NATs, and it includes a server initiated call-back to the client as part of the communication protocol. This cannot take place if the client is behind a NAT, requiring a complete rewrite of the NetSolve system. Additionally, NetSolve keeps track of components by IP addresses, which are not globally unique in the presence of private subnets managed by NATs.

NetSolve also uses a wide range of ports for its communications. In this current era of increased network security and omnipresent firewalls, this requirement was awkward to meet. Many sites with strong firewall policies are not setup to unblock and allow network traffic on a wide range of ports.

From the beginning, NetSolve was designed to make it possible for users to add additional services to their servers, to allow them to turn their custom

applications into services that can be executed on powerful remote platform and can be accessed easily from desktop clients. However, experience has shown that this process was too complicated for many users, since adding services to NetSolve requires preparing a fairly idiosyncratic service description file, which uses mnemonic fields to describe data types and structure.

When a service is added to NetSolve, a measure of the computational complexity of the service needs to be provided to make it possible for the NetSolve agent to estimate the execution time of that service on various servers and thus rank the servers. This computational complexity was described using a minimal model, which makes it difficult to implement more complex and accurate scheduling algorithms.

3 GridSolve: A Network Enabled Solver

The GridSolve project is an evolution of NetSolve, architected to overcome the shortcomings of NetSolve and to provide a platform for additional development and experimentation. The system architecture of GridSolve is the same as that of NetSolve shown in Figure 1, where a client-agent-server system interact to provide transparent Grid based services to an end-user.

The overall goal of the GridSolve project is to address three general problems: ease of use, interoperability, and extensibility. Improving ease of use refers to improving the process by which libraries and services are added into a GridSolve server. Interoperability encompasses several facets, including better handling of different network topologies, and better interaction with other Grid computing projects. Extensibility in this context means easy extension to new parallel libraries and architectures, support for large datasets, and better resource scheduling to take advantage of growing set of servers and services.

3.1 Ease of use

IDL improvements One of the original design goals was to eliminate the need for client-side stubs for each procedure in a remote procedure call (RPC) environment. However, this design decision tends to push the complexity to the servers. Integrating new software into NetSolve required writing a complex server side interface definition (Problem Description File), which specifies the parameters, data types, and calling sequence. Despite several attempts to create a user-friendly tool to generate the Problem Description Files, it can still be a difficult and error-prone process.

Therefore, we have implemented a simple technique for adding additional services to a running GridSolve server. The interface definition format itself has been greatly simplified and the services are compiled as external executables with interfaces to the server described in a standard format. The server re-examines its own configuration and installed services periodically or when

it receives the appropriate signal. In this way it becomes aware of any additional services that are installed without re-compilation or restarting. The server reports the new service to the agent, and thereafter it can be used by any GridSolve client.

3.2 Interoperability

Handling NATs A Network Address Translator [8] presents the same external IP address for all machines within a private subnet, reducing the overall need for unique IP addresses. NATs are often used by end-users as a way of providing multiple machines with network access without requiring that they all be assigned unique global IP addresses. They are also sometimes used a security measure since it is difficult to make inbound connectivity to a machine behind a NAT. However, this causes problems for services such as GridSolve such as: IP addresses may not be unique, IP address-to-host bindings may not be stable, and hosts behind the NAT may not be contactable from outside. To address these issues we have developed a new communications framework for GridSolve. To avoid problems related to potential duplication of IP addresses, the GridSolve components are identified by a globally unique identifier specified by the user or generated randomly. To allow inbound connectivity to GridSolve servers behind a NAT, a GridSolve *proxy* executable is distributed with the software. If enabled, a GridSolve server will use the proxy to channel all communications, keeping a connection to the proxy open at all times. This makes the server usable by clients that would not have been able to connect to the server otherwise.

Firewall concerns To handle firewalls in a more adaptive manner, GridSolve now restricts itself to specific ports for communication. The ports can be specified in the execution environment, allowing communication over any port, including the default HTTP port if necessary, since this port are almost always setup to allow traffic through a firewall.

GridRPC API The GridRPC API was made the core API for GridSolve, enabling compatibility with other Grid programming efforts such as Ninf-G or DIET. Additional capabilities such as the Matlab API are built on top of the GridRPC API. The older NetSolve API is also build on top of the GridRPC API to allow backward compatibility for users that did development using NetSolve.

3.3 Extensibility

Supporting backend resource managers In the older NetSolve system, backend resource and execution managers such as Condor and OpenPBS were supported by creating a specialized server for that environment and compiling it into the server. Though effective, this method was cumbersome and required knowledge of the internals of the code. In GridSolve, supporting different backends has

been made easier by defining a interface that requires three scripts for service initiation, probing and cancellation. These scripts are specified within the service description, easily allowing any library routine to be run either on a backend or directly on a GridSolve server.

Disconnected Operation Since some of the backend resource managers (e.g., batch queues) may take a substantial time to execute an application, GridSolve has been extended to support disconnected operations. After a GridSolve service request has been submitted asynchronously, the user can request a serialized representation of the service request. This can be saved, and then used to return to the service at a later time.

Scheduling enhancements GridSolve will retain the familiar agent-based scheduling of resources [13], but in some cases the client has additional knowledge about the appropriate set of resources. Therefore we are implementing an infrastructure that allows resource filtering to be optionally performed by the client. In the older NetSolve system, the only user-provided filter that affects the selection of resources is the problem name. Given the problem name, the agent filters the available servers to select the those that can solve that problem, and then ranks the servers. In the newer GridSolve system, the user can provide additional constraints on the filtering process, for example, a minimum memory requirement or the availability of a database. Also, the client will have access to the complete list of resources and their characteristics so that the user can implement comprehensive scheduling algorithms in addition to enhanced filtering. To enable this functionality, a GridSolve server should provide as much information as possible to the agent as free-form resource attributes. The agent then uses the resource attributes to match the filtering request of the client.

Distributed Storage Infrastructure GridSolve supports a Distributed Storage Infrastructure (DSI) API, allowing it to deal with large data in a efficient manner. Using DSI, a client can deploy large data items, such as a vector or matrix, into high speed network storage. Then, when calling a service, a handle to the data can be transparently provided instead of the data item itself. This allows the service to access the data quickly, and the service can reuse the data from the network storage rather than fetching it from the client on each use. This style of deployment could also allow the user to handle data that is too large to fit into the memory of their local computer. Currently, DSI is implemented on top of the Internet Backplane Protocol (IBP) [3] which provides middleware for managing and using remote storage.

4 Related Work

Several Network Enabled Servers (NES) provide mechanisms for transparent access to remote resources and software. Ninf-G [12] is a reference implementation of the GridRPC API [11] built on top of the Globus Toolkit. Ninf-G provides

an interface definition language that allows services to be easily added, and client binding are available in C and Java. Security, scheduling and resource management are left up to Globus.

The DIET (Distributed Interactive Engineering Toolbox) project [4] is a client-agent-server RPC architecture which uses the GridRPC API as its primary interface. A CORBA Naming Service handles the resource registration and lookup, and a hierarchy of agents handles the scheduling of services on the resources. An API is provided for generating service profiles and adding new services, and a C client API exists.

NEOS [7] is a network-enabled problem-solving environment designed as a generic application service provider (ASP). Any application that can be changed to read its inputs from files, and write its output to a single file can be integrated into NEOS. The NEOS Server acts as an intermediary for all communication. The client data files go to the NEOS server, which sends the data to the solver resources, collects the results and then returns the results to the client. Clients can use email, web, sockets based tools and CORBA interfaces.

Other projects are related to various aspects of GridSolve. For example, task farming style computation is provided by the Apples Parameter Sweep Template (APST) project [6], the Condor Master Worker (MW) project [10], and the Nimrod-G project [1]. Request sequencing is handled by projects like Condor DAGman [9].

However, GridSolve provides a complete solution for easy access to remote resources and software. It differs from the other NES implementations by including a tight, simple integration with client PSEs such as Matlab. Interface descriptions for a variety of standard mathematical libraries are distributed with GridSolve, and it is easy for additional services to be added. The ability to use server-proxies to make it possible to leverage additional resource management and scheduling environments also adds to GridSolve's strengths.

5 Ongoing Work and Conclusion

GridSolve is still in an early release phase, as it has not yet implemented all the functionality of its predecessor NetSolve. Some of the ongoing work in the GridSolve project is described below.

- Currently the Matlab client bindings are available, and there is some work done on generating client bindings for IDL (Interactive Data Language). Additional languages such as Mathematica, Octave and Java still need to be added.
- A small set of library bindings is currently distributed with GridSolve (i.e., a subset of LAPACK and SuperLU). A more complete set of libraries bindings (LAPACK, ScaLAPACK, SuperLU, ARPACK and PETSc) will be added.
- There is a Kerberos based security mechanism in the current GridSolve distribution. We are investigating other possibilities to enable better integration with additional security infrastructures.

– Ongoing research is investigating ways to use the history of service executions to build an execution model for the services. These models are then used in a more accurate scheduling of the services on servers.
– Since the GridSolve agent currently maintains information about all resources in the entire system, it may be a scalability bottleneck as the number of resources increases. We are investigating the use of multiple cooperating agents to allow the GridSolve system to be scalable.

The GridSolve project has been designed to fit the needs of the general community of scientists and engineers, to provide an easy to use interface to Grid hardware and software resources. A GridSolve user is relieved of many of the details that make using Grid resources awkward: finding the appropriate resources, ensuring that the needed libraries are installed, submitting the application to the resources, monitoring the execution of the application and transferring results back to their SCE for further viewing and analysis.

The current version GridSolve incorporates major enhancements that are based on real world experience and user feedback. These enhancements include tolerance for NATs, accelerated performance, disconnected operation, improved service setup and deployment, resource filtering and improved scheduling.

References

1. David Abramson, Rajkumar Buyya, and Jonathan Giddy. A computational economy for Grid Computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems*, 18(8):1061–1074, October 2002.
2. D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4.1. Innovative Computing Laboratory. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.
3. A. Bassi, M. Beck, T. Moore, J. Plank, M. Swany, R. Wolski, and G. Fagg. The Internet Backplane Protocol: A Study in Resource Sharing. In *Future Generation Computing Systems*, volume 19, pages 551–561.
4. E. Caron, F. Desprez, F. Lombard, J.-M. Nicod, L. Philippe, M. Quinson, and F. Suter. A scalable approach to network enabled servers (research note). *Lecture Notes in Computer Science*, 2400, 2002.
5. Henri Casanova and Jack Dongarra. NetSolve: A Network-Enabled Server for Solving Computational Science Problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, Fall 1997.
6. Henri Casanova, Graziano Obertelli, Berman Berman, and Rich Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proceedings of Supercomputing'2000 (CD-ROM)*, Dallas, TX, Nov 2000. IEEE and ACM SIGARCH.
7. E. Dolan, R. Fourer, J. J. Moré, and Munson Munson. The NEOS server for optimization: Version 4 and beyond. Technical Report ANL/MCS-P947-0202, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, February 2002.

8. K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631, May 1994.
9. James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steve Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, 2002.
10. Jeff Linderoth, Sanjeev Kulkarni, Jean-Pierre Goux, and Michael Yoder. An Enabling Framework for Master-Worker Applications on the Computational Grid. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pages 43–50, Pittsburgh, PA, August 2000.
11. K. Seymour, N. Hakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova. Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In M. Parashar, editor, *GRID 2002*, pages 274–278, 2002.
12. Y. Tanaka, H. Nakada, S. Sekiguchi, Suzumura Suzumura, and S. Matsuoka. Ninf-G: A reference implementation of RPC-based programming middleware for Grid computing. *Journal of Grid Computing*, 1(1):41–51, 2003.
13. Asim YarKhan, Keith Seymour, Kiran Sagi, Zhiao Shi, and Jack Dongarra. Recent Developments in Gridsolve. *International Journal of High Performance Computing Applications (IJHPCA)*, 20(1):131–141, 2006.