Overview of GridRPC: A Remote Procedure Call API for Grid Computing *

Keith Seymour[1], Hidemoto Nakada[5,6], Satoshi Matsuoka[6,7], Jack Dongarra[1], Craig Lee[4], and Henri Casanova[2,3]

[1] Department of Computer Science, University of Tennessee, Knoxville
[2] San Diego Supercomputer Center
[3] Dept. of Computer Science and Engineering, University of California, San Diego
[4] Computer Systems Research Dept., The Aerospace Corp., El Segundo, California
[5] National Institute of Advanced Industrial Science and Technology (AIST)
[6] Tokyo Institute of Technology
[7] National Institute of Informatics

Abstract. This paper discusses preliminary work on standardizing and implementing a remote procedure call (RPC) mechanism for grid computing. The GridRPC API is designed to address the lack of a standardized, portable, and simple programming interface. Our initial work on GridRPC shows that client access to existing grid computing systems such as NetSolve and Ninf can be unified via a common API, a task that has proven to be problematic in the past.

1 Introduction

Although Grid computing is regarded as a viable next-generation computing infrastructure, its widespread adoption is still hindered by several factors, one of which is the question "how do we program on the Grid (in an easy manner)". By all means there have been various attempts to provide a programming model and a corresponding system or a language appropriate for the Grid. Many such efforts have been collected and catalogued by the Advanced Programming Models Research Group of the Global Grid Forum [1]. One particular programming model that has proven to be viable is an RPC mechanism tailored for the Grid, or "GridRPC". Although at a very high level view the programming model provided by GridRPC is that of standard RPC plus asynchronous coarse-grained parallel tasking, in practice there are a variety of features that will largely hide the dynamicity, insecurity, and instability of the Grid from the programmers. As such, GridRPC allows not only enabling individual applications to be distributed, but also can serve as the basis for even higher-level software substrates such as distributed, scientific components on the Grid. Moreover, recent work [2] has shown that GridRPC could be effectively built upon future Grid software based on Web Services such as OGSA [3].

* This work funded in part by a grant from the NSF EIA-9975015.

This work was motivated by earlier attempts to achieve interoperability between by two systems that provide *network-enabled services*, namely NetSolve [4] and Ninf [5]. This proved to be difficult, indicating the need for a more unified effort to understand the requirements of a GridRPC model and API. This is reported in the next section.

2 The GridRPC Model and API

Function Handles and Session IDs. Two fundamental objects in the GridRPC model are *function handles* and the *session IDs*. The function handle represents a mapping from a function name to an instance of that function on a particular server. Once a particular function-to-server mapping has been established all RPC calls using that function handle will be executed on the server specified in that binding. A session ID is an identifier representing a particular non-blocking RPC call.

Initializing and Finalizing Functions. The initialize and finalize functions are similar to the MPI initialize and finalize calls. Client GridRPC calls before initialization or after finalization will fail.

- `grpc_initialize` reads the configuration file and initializes the modules.
- `grpc_finalize` releases any resources being used by GridRPC.

Remote Function Handle Management Functions. The *function handle management* group of functions allows creating and destroying function handles.

- `grpc_function_handle_default` creates a handle using the default server.
- `grpc_function_handle_init` creates a handle with a user-specified server.
- `grpc_function_handle_destruct` frees the memory for the specified handle.
- `grpc_get_handle` returns the handle corresponding to the given session ID.

GridRPC Call Functions. The four GridRPC call functions may be categorized by a combination of two properties: blocking behavior and calling sequence. A call may be either blocking (synchronous) or non-blocking (asynchronous) and it may use either a variable number of arguments (like `printf`) or an *argument stack* calling sequence (see Section 2).

- `grpc_call` makes a blocking call (variable argument list).
- `grpc_call_async` makes a non-blocking call (variable argument list).
- `grpc_call_argstack` makes a blocking call (argument stack).
- `grpc_call_argstack_async` makes a non-blocking call (argument stack).

Asynchronous GridRPC Control and Wait Functions. The following functions allow probing the status or waiting for completion of previously submitted non-blocking requests. The wait calls allow an application to express desired non-deterministic completion semantics to the underlying system, rather than repeatedly polling on a set of sessions IDs.

- `grpc_probe` checks whether the asynchronous GridRPC call has completed.
- `grpc_cancel` cancels the specified asynchronous GridRPC call.
- `grpc_wait` blocks until the specified non-blocking request completes.
- `grpc_wait_and` waits for *all* of the non-blocking requests in a given set.
- `grpc_wait_or` waits for *any* of the non-blocking requests in a given set.
- `grpc_wait_all` waits for *all* previously issued non-blocking requests.
- `grpc_wait_any` waits for *any* previously issued non-blocking request.

Error Reporting Functions. The following error reporting functions provide error codes and human-readable error descriptions.

- `grpc_perror` prints the error string associated with the last GridRPC call.
- `grpc_error_string` returns the error description string, given an error code.
- `grpc_get_error` returns the error code for a given non-blocking request.
- `grpc_get_last_error` returns the error code for the last invoked call.

Argument Stack Functions. With the following argument stack interface it is possible to construct the arguments to a function call at run-time.

- `grpc_arg_stack_new` creates a new argument stack.
- `grpc_arg_stack_push_arg` pushes the specified argument onto the stack.
- `grpc_arg_stack_pop_arg` removes the top element from the stack.
- `grpc_arg_stack_destruct` frees the memory for the specified argument stack.

3 Implementations

3.1 GridRPC over NetSolve

NetSolve [4] is a client-server system which provides remote access to hardware and software resources through a variety of client interfaces, such as C, Fortran, and Matlab. Since NetSolve's mode of operation is in terms of RPC-style function calls, it provides much of the infrastructure needed to implement GridRPC.

Overview of NetSolve. A NetSolve system consists of three entities, as illustrated in Figure 1. The *Client*, which requests remote execution of some function (through C, Fortran, or an interactive program such as Matlab or Mathematica). The *Server* executes functions on behalf of the clients. The server hardware can range in complexity from a uniprocessor to a MPP system and similarly the functions executed by the server can be arbitrarily complex. The *Agent* is the focal point of the NetSolve system. It maintains a list of all available servers and performs resource selection for all client requests as well as ensuring load balancing of the servers.

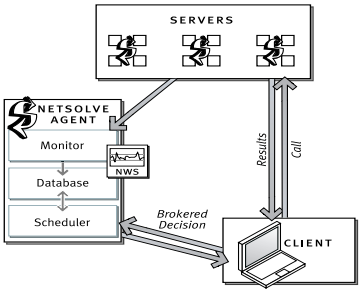

Fig. 1. Overview of NetSolve.

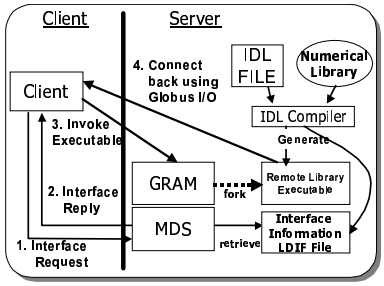

Fig. 2. Overview of Ninf-G.

Using NetSolve to Implement GridRPC. Currently we have a full implementation of the GridRPC API running on top of the NetSolve system. An important factor in enabling the implementation of GridRPC in NetSolve is the strong similarity of their APIs. Besides this similarity, NetSolve has several properties that make it an attractive choice for implementing GridRPC: fault-tolerance, load-balancing, and security.

3.2 GridRPC over Ninf

Overview of Ninf-G. Ninf-G is a re-implementation of the Ninf system [5] on top of the Globus Toolkit. The Globus toolkit provides a reference implementation of standard (or subject to proposed standardization) protocols and APIs for Grid computing. Globus serves as a solid and common platform for implementing higher-level middleware and programming tools, etc., ensuring interoperability amongst such high-level components, one of which is Ninf-G. Figure 2 shows an overview of the Ninf-G system in this regard.

Ninf-G is designed focusing on simplicity. In contrast with NetSolve, Ninf-G does not provide fault detection, recovery or load-balancing by itself. Instead, Ninf-G assumes that backend queuing system, such as Condor, takes responsibility for these functionality. Ninf-G fully deploys Globus Security Infrastructure. It means that not only all the components are protected properly, but also they can utilize other Globus components, such as GridFTP servers, seamlessly and securely.

Using Ninf-G to Implement GridRPC. As in NetSolve, the Ninf-G design allows direct support for the GridRPC model and API. The steps in making an actual Ninf-G GridRPC call can be broken down into those shown in Figure 2.

1. *Retrieval of interface information and executable pathname.*
2. *MDS sends back the requested information.*
3. *Invoking remote executable.*
4. *Remote executable callbacks to the client.*

4 Related Work and Conclusions

We have presented preliminary work in defining a model and API for a grid-aware RPC mechanism. The concept of Remote Procedure Call (RPC) has been widely used in distributed computing and distributed systems for many years. Previous work in this area has focused on RPC mechanisms for single processors, tightly-coupled homogeneous processors, and also distributed objects, such as CORBA and Java RMI. The work reported here focuses on RPC functionality that meets the needs of scientific computing among loosely-coupled heterogeneous systems over wide-area networks, while allowing multiple implementations. While the NetSolve and Ninf implementations are reported here, the GridRPC API does not preclude the internal use of XML-based protocols since this is not exposed through the API. Since a more complete discussion is beyond the scope of this short paper, the interested reader is referred to [6].

While the model and API presented here is a first-step towards a general GridRPC capability, there are certainly a number of outstanding issues regarding wide-spread deployment and use. Quite briefly these include *discovery, meta-data schemas, scheduling* (including the *co-scheduling* of multiple GridRPCs), and *workflow management* among multiple servers. Transitive and composible *fault-tolerance* and *security* will also have to be provided across a potentially distributed call-tree of GridRPCs. The development of a practical, basic GridRPC capability, however, will produce a body of experience that will establish the priorities for such future work.

References

1. C. Lee, S. Matsuoka, D. Talia, A. Sussman, M. Mueller, G. Allen, and J. Saltz. A Grid Programming Primer. `http://www.gridforum.org/7_APM/APS.htm`, submitted to the Global Grid Forum, August 2001.
2. Satoshi Shirasuna, Hidemoto Nakada, Satoshi Matsuoka, and Satoshi Sekiguchi. Evaluating Web Services Based Implementations of GridRPC. In *Proc. of HPDC11*, pages 237–245, 2002.
3. Ian Foster, Carl Kesselman, Jeffrey Nick, and Steven Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. `http://www.globus.org/ogsa`, January 2002.
4. D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4. Computer Science Dept. Technical Report CS-01-467, University of Tennessee, Knoxville, TN, July 2001. See also `http://icl.cs.utk.edu/netsolve`.
5. Hidemoto Nakada, Mitsuhisa Sato, and Satoshi Sekiguchi. Design and Implementations of Ninf: towards a Global Computing Infrastructure. In *Future Generation Computing Systems, Metacomputing Issue*, volume 15, pages 649–658, 1999. See also `http://ninf.apgrid.org`.
6. K. Seymour, N. Hakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova. GridRPC: A Remote Procedure Call API for Grid Computing (long version). `http://www.eece.unm.edu/~apm/docs/APM_GridRPC_0702.pdf`, July 2002.