GWD-R
Category: Recommendation
GridRPC Working Group

H. Nakada, NIAIST
S. Matsuoka, Tokyo Institute of Tech.
K. Seymour, Univ. of Tenn., Knoxville
J. Dongarra, Univ. of Tenn., Knoxville
C. Lee, The Aerospace Corp.
H. Casanova, UCSD, SDSC
July 29, 2004

A GridRPC Model and API for Advanced and Middleware Applications

Abstract

This document presents advanced features of a model and API for GridRPC, i.e., a remote procedure call (RPC) mechanism for grid environments. Specifically this document is targeted for applications by advanced users and middleware developers. This document cannot be fully understood in isolation; it must be considered in conjunction with *A GridRPC Model and API for End-User Applications* which documents the fundamental concepts for this GridRPC model. This document extends the End-User API in the following ways. (1) Variable length array arguments are defined to be used in middleware libraries based on GridRPC. (2) Call attribute introspection is supported where arguments and other GridRPC elements are first class objects that can be managed. (3) Support for *persistent data* and *workflow management* is provided that exposes just enough semantics in the API to be able to deal with underlying grid workflow engines.

1. Introduction

The goal of this document is to clearly and unambiguously extend the syntax and semantics of GridRPC, a remote procedure call (RPC) mechanism for grid environments, thereby providing support for advanced applications and middleware libraries based on GridRPC. This document cannot be fully understood in isolation; it must be read an understood in conjunction with *A GridRPC Model and API for End-User Applications* [14] which documents the fundamental concepts for this GridRPC model. A preliminary version of that document appeared as [17]. A longer version is available as [18].

This document extends the End-User API in the three following ways: (1) variable length array arguments, (2) call attribute introspection, and (3) support for *persistent data* and *workflow management*. These capabilities are defined in the next three sections.

2. Variable Length Array Arguments

2.1 Additional GridRPC Data Types

grpc_arg_stack_t

This data type is used for argument stacks.

2.2 Argument Stack/Vector Functions

*Comment, CAL: Should this be a stack or a vector? Depending on this, we will need to change the push and pop calls below. Should we define a "grpc_arg *" to be used instead of "void *"?*

These functions are used to construct a stack of arguments at runtime. When interpreted as a list of arguments, the stack is ordered from bottom up. That is, to emulate a function call **f(a,b,c)**, the user would push the arguments in the same order: **push(a); push(b); push(c);**. Note that an argument stack has a fixed size when it is allocated. It does not, however, have to be "fully populated" when used as a call argument. Only those arguments on the stack at the time of the call will be used.

grpc_error_t *grpc_stack_init(grpc_arg_stack_t *stack, int maxsize)

This initializes a new argument stack. *maxsize* is the maximum number of arguments that can be pushed on to this stack.

Comment, CAL: Error table needs to be corrected.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_stack_push(grpc_arg_stack_t *stack, void *arg)

This pushes the specified argument onto the stack. If this push operation exceeds the size of the stack argument, the stack is not changed and an error is returned.

Comment, CAL: Error table needs to be corrected.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_stack_pop(grpc_arg_stack_t *stack, void ** arg)

This removes the top element from the stack and returns it as the argument. If the stack is empty, a null pointer is returned in **arg**.

Comment, CAL: Error table needs to be corrected.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_stack_destruct(grpc_arg_stack_t *stack)

This frees all content associated with the specified argument stack.

Comment, CAL: Error table needs to be corrected.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_OTHER_ERROR_CODE	Internal error detected

2.3 Stack/Vector-based GridRPC Call Functions

The argument stack calling sequence allows building the list of arguments to the function at runtime through elementary stack operations, such as *push* and *pop*. Like the end-user call, both synchronous and asynchronous versions are available.

Rationale:

The use of stack arguments allows the GridRPC API to be used in middleware when the number of arguments is not known at compile-time. Stack arguments can be seen as more fundamental since all remote execution calls could be expressed as *apply(func_name, arg_stack)*. This can enable type-checking and is more portable than varargs. Allowing a variable number of arguments is, nonetheless, a tremendous convenience for end-users and a very common practice.

End of Rationale.

grpc_error_t grpc_call_arg_stack(grpc_function_handle_t *handle, grpc_arg_stack_t *args)

This makes a blocking call using the argument stack.

Comment, CAL: Error table needs to be corrected.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_call_arg_stack_async(
grpc_function_handle_t *handle,
grpc_sessionid_t *sessionID,
grpc_arg_stack_t *args)

This makes a non-blocking call using the argument stack. Similarly, a *session ID* is returned that can be used to probe or wait for completion, cancel the call, and check for the error status of a call.

Comment, CAL: Error table needs to be corrected.

Additional Error Code Identifiers	Meaning
GRPC_STACK_NOT_INITIALIZED	Argument stack/vector variable not initialized before use
GRPC_STACK_OVERFLOW	Argument stack overflowed as a result of one too many pushes
GRPC_EXCEED_LIMIT	
GRPC_INVALID_LENGTH	

Table 1. Summary of Additional GridRPC Error Codes

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_OTHER_ERROR_CODE	Internal error detected

As with the end-user calls, this document does not define which implementation-related operations may be assumed to be complete when an asynchronous call returns. However, all asynchronous GridRPC calls must return as soon as possible after it is safe for a user to modify any input argument buffers.

2.4 Additional Error Codes

Comment, CAL: Error table needs to be corrected.

Table 1 gives the additional error code identifiers related to stack/vector arguments that can be used with variables of type *grpc_error_t*. These error codes are generated and used in the same way as error codes in the End-User document. These error codes satisfy:

$$0 = \text{GRPC\_NO\_ERROR} < \text{GRPC\_...} < \text{GRPC\_LAST\_ERROR\_CODE}$$

3. Call Attribute Introspection

Comment, CAL: Included here verbatim is the text of an email from Satoshi Matsuoka that oddly enough does not appear in the GridRPC GGF email archive but I have in my GridRPC mail folder (and clearly remember from when it was sent).

Comment, SM:

```
My two cent's metacomment on the issue:


Hidemoto's request is not merely particular or accidental, but rather
is an inevitable consequence of design undertaking providing a
higher-level procedure call (or message passing) interface based on a
lower-level one.


Indeed this has been the case for abundunce of past research on remote
invocations and distributed object message passing, where remote calls
are intercepted and new features would be added; there, it has been
largely agreed that systematic introspective features of call attributes
including the call arguments embodied in some non-opaque structured
datatype is absolutely necessary. This has carried on to more commercial
implementations as a part of CORBA (Dynamic Invocation Interface) and
```

```
Java RMI (with reflection API), and GridRPC would be no exception. We
must recognize and follow suite, as a principle for constructing
interfaces of this sort.


That is to say we should think of making (certain parts of) RPC call
elements to be first class objects, such as:


        Sender, Receiver
        Arguments (and their types)
        other attributes, such as
                RPC sender call site (i.e., location within the program)
                Various network attributes of the call
                connection method
                # of retries
                etc.etc.


Not all of them are required; rather, we should think of which would be
relevant for GridRPC, and higher-level APIs we intend to implement above
it. So far we have found that making the set of arguments as some first-class
manipulable object is essential for implementing a fault-tolerant,
farming extension of the GridRPC API on top of the current GridRPC API.


                                                        Satoshi
```

4. Persistent Data and Workflow Management

Comment, CAL: We have several ideas of how to manage persistent data and workflow using the concept of data handles. While we shouldn't really do generalized workflow, we should expose just enough through the API such that an advanced GridRPC user can manage and interact with any number of grid workflow engines. Much work to do in this section.

5. Related Work

Comment, CAL: This is just copied from the End-User doc. It really should be tailored to work that is related to building middleware libraries, introspection, and workflow management.

The concept of Remote Procedure Call (RPC) has been widely used in distributed computing and distributed systems for many years [4]. It provides an elegant and simple abstraction that allows distributed components to communicate with well-defined semantics. RPC implementations face a number of difficult issues, including the definition of appropriate Application Programming Interfaces (APIs), wire protocols, and Interface Description Languages (IDLs). Corresponding implementation choices lead to trade-offs between flexibility, portability, and performance.

A number of previous works has focused on the development of high performance RPC mechanisms either for single processors or for tightly-coupled homogeneous parallel computers such as shared-memory multiprocessors [7, 3, 13, 2]. A contribution of those works is to achieve high performance by providing RPC mechanisms that map directly to low-level O/S and hardware functionalities (e.g. to move away from implementations that were built on top of existing message passing mechanisms as in [5]). By contrast, GridRPC targets heterogeneous and loosely-coupled systems over wide-are networks, raising a different set of concerns and goals.

This current work grew out of the Advanced Programming Models Research Group [10]. This group surveyed and evaluated many programming models [11, 12], including GridRPC. Some representative GridRPC systems are NetSolve [6], and Ninf [15]. Historically, both projects started about the same time, and in fact both systems facilitate similar sets of features. A number of related experimental systems exist, such as RCS [1] and Punch [16]. Those systems seek to provide ways for Grid users to easily send requests to remote application servers from their desktop. GridRPC seeks to unify those efforts.

This work is also related to the XML-RPC [21] and SOAP [20] efforts. Those systems use HTTP to pass XML fragments that describe input parameters and retrieve output results during RPC calls. In scientific computing, parameters to RPC calls are often large arrays of numerical data (e.g. double precision matrices). The work in [9] made it clear that using XML encoding has several caveats for those types of data (e.g. lack of floating-point precision, cost of encoding/decoding). Nonetheless, recent work [19] has shown that GridRPC could be effectively built upon future Grid software based on Web Services such as OGSA [8].

6. Security

Security issues are not discussed in this document.

7. Author Contact Information

Hidemoto Nakada
Natl. Inst. of Advanced Industrial Science and Technology
hide-nakada@aist.go.jp

Satoshi Matsuoka
Tokyo Institute of Technology
National Institute of Informatics
matsu@is.titech.ac.jp

Keith Seymour
Univ. of Tennessee, Knoxville
seymour@cs.utk.edu

Jack Dongarra
Univ. of Tennessee, Knoxville
dongarra@cs.utk.edu

Craig A. Lee
The Aerospace Corporation, M1-102
2350 E. El Segundo Blvd.
El Segundo, CA 90245
lee@aero.org

Henri Casanova
University of California, San Diego
San Diego Supercomputing Center
casanova@cs.ucsd.edu

Intellectual Property Statement

which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director.

Full Copyright Notice

References

[1] P. Arbenz, W. Gander, and M. Oettli. The Remote Computation System. *Parallel Computing*, 23:1421–1428, 1997.

[2] I. Aumage, L. Boug, A. Denis, J.-F. Mhaut, G. Mercier, R. Namyst, and L. Prylli. Madeleine II: A Portable and Efficient Communication Library for High-Performance Cluster Computing. In *Proceedings of the IEEE Intl Conference on Cluster Computing (Cluster 2000)*, pages 78–87, 2000.

[3] B. Bershad, T. Anderson, E. Lazowska, and H. Levy. Lightweight Remote Procedure Call. *ACM Transactions on Computer Systems (TOCS)*, 8(1):37–55, 1990.

[4] A. Birrel and G. Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems (TOCS)*, 2(1):39–59, 1984.

[5] L. Boug, J.-F. Mhaut, and R. Namyst. Efficient Communications in Multithreaded Runtime Systems. In *Proceedings of the 3rd Workshop on Runtime Systems for Parallel Programming (RTSPP'99), volume 1568 of Lecture Notes in Computer Science, Springer Verlag*, pages 468–484, 1999.

[6] H. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. In *Proceedings of Super Computing '96*, 1996.

[7] C.-C. Chang, G. Czajkowski, and T. von Eicken. MRPC: A High Performance RPC System for MPMD Parallel Computing. *Software - Practice and Experience*, 29(1):43–66, 1999.

[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www.globus.org/ogsa, January 2002.

[9] M. Govindaraju, A. Slominski, V. Choppella, R. Bramley, and D. Gannon. Requirements for and Evaluation of RMI Protocols for Scientific Computing. In *Proceedings of SC'2000, Dallas, TX*, 2000.

[10] Grid Forum Advanced Programming Models Working Group. Web site. *http://www.eece.unm.edu/~apm*, 2000.

[11] C. Lee, S. Matsuoka, D. Talia, A. Sussman, M. Mueller, G. Allen, and J. Saltz. A Grid Programming Primer. http://www.eece.unm.edu/~apm/docs/APM_Primer_0801.pdf, August 2001.

[12] C. Lee and D. Talia. Grid programming models: Current tools, issues and directions. In Berman, Fox, and Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, pages 555–578. Wiley, 2003.

[13] J. Liedtke. Improving IPC by Kernel Design. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles (SOSP), Asheville, NC*, Dec. 1993.

[14] H. Nakada et al. A GridRPC Model and API for End-User Applications. https://forge.gridforum.org/projects/gridrpc-wg/document/GridRPC_EndUser_16dec03/en/1, December 2003.
[15] H. Nakada, M. Sato, and S. Sekiguchi. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15(5-6):649–658, 1999.
[16] The Punch project at Purdue. http://punch.ecn.purdue.edu.
[17] K. Seymour et al. An Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In *3rd International Workshop on Grid Computing*, volume 2536, pages 274–278. Springer-Verlag, Lecture Notes in Computer Science, November 2002.
[18] K. Seymour et al. GridRPC: A Remote Procedure Call API for Grid Computing. http://www.eece.unm.edu/~apm/docs/APM_GridRPC_0702.pdf, July 2002.
[19] S. Shirasuna, H. Nakada, S. Matsuoka, and S. Sekiguchi. Evaluating Web Services Based Implementations of GridRPC. In *Proc. of HPDC11*, pages 237–245, 2002.
[20] Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/SOAP, May 2000. W3C Note.
[21] XML-RPC. http://www.xml-rpc.com.

Contents