GFD-R.052
Category: Recommendation-modified
GridRPC Working Group

H. Nakada, AIST
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

September 23, 2004
February 17, 2005
July 21, 2005
April 12, 2007

A GridRPC Model and API for End-User Applications

Status of This Memo

This document provides a recommendation to the Grid community on a proposed model and API for a grid-enabled remote procedure call. Distribution is unlimited.

Copyright Notice

Abstract

This document presents a model and API for GridRPC, i.e., a remote procedure call (RPC) mechanism for grid environments. Specifically this document is targeted for end-user applications, not middleware. That is to say, this document presents a simpler version of the GridRPC model and API that is completely sufficient for end-users and does not include the more complex features and capabilities required for building middleware. As a Recommendations track document in the Global Grid Forum, the goal of this document is to clearly and unambiguously define the syntax and semantics for GridRPC, thereby enabling a growing user base to take an advantage of multiple implementations. The motivation for this document is to provide an easy avenue of adoption for grid computing, since (1) RPC is an established distributed computing paradigm, and (2) there is a growing user-base for network-enabled services. By doing so, this document will also facilitate the development of multiple implementations.

Contents

Abstract....................1
1. Introduction....................2
2. The Basic GridRPC Model....................2
3. Document Scope....................2
3.1 In Scope....................2
3.2 Out of Scope....................3
4. The GridRPC API....................3
4.1 GridRPC Data Types....................3
4.2 Initializing and Finalizing Functions....................4
4.3 Remote Function Handle Management Functions....................4
4.4 GridRPC Call Functions....................6
4.5 Asynchronous GridRPC Control Functions....................7
4.6 Synchronous GridRPC Wait Functions....................8
4.7 Error codes and Error Reporting Functions....................10
5. Related Work....................12
6. Security Considerations....................12
Author Contact Information....................12
Intellectual Property Statement....................13
Full Copyright Notice....................13
References....................14

1. Introduction

The goal of this document is to clearly and unambiguously define the syntax and semantics for GridRPC, a remote procedure call (RPC) mechanism for grid environments, thereby providing an avenue of easy access to grid computing. Specifically this document is targeted for end-user applications that do not require the more complex features and capabilities required for middleware packages. As such, it is outside the scope of this document to review or discuss those issues related to middleware, or the important issues related to network-enabled services or to provide any kind of tutorial information. Nonetheless, a Related Work section is provided to capture many references and pointers to relevant works that have lead up to this document. A preliminary version of this model and API appeared as [15]. A longer version of that paper is available as [16]. Comparison with CORBA is shown as [21].

2. The Basic GridRPC Model

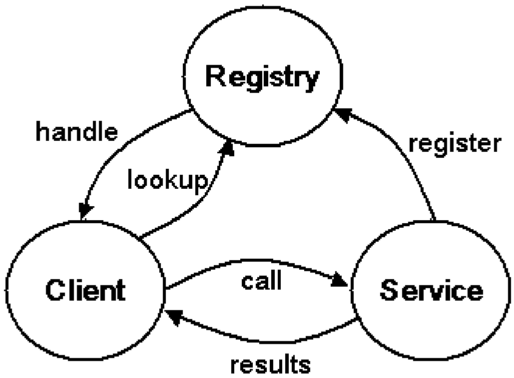

Figure 1. The Basic GridRPC Model.

Figure 1 illustrates the basic GridRPC model. The functions shown here are very fundamental and, hence, appear in many other systems. A service registers with a registry. A client subsequently contacts the registry to look-up a desired service and the registry returns a handle to the client. The client then uses the handle to call the service which eventually returns the results.

In the GridRPC terminology adopted here, the service handle is a *function handle* which represents a mapping from a simple, flat function name string to an instance of that function on a particular server. Once a particular function-to-server mapping has been established by initializing a *function handle*, 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 GridRPC call. The *session ID* is used throughout the API to allow users to obtain the status of a previously submitted non-blocking call, to wait for a call to complete, to cancel a call, or to check the error code of a call.

3. Document Scope

This simple, common model nonetheless represents multiple fundamental issues. It is clearly impossible to deal with them all at the same time. Hence, we now clarify what this document defines and does not define.

3.1 In Scope

This document focuses on just defining the API and the minimal programming model needed to understand and use the API for end-user applications. More specifically, it focuses on simple, client-server interaction since this comprises the majority of usage scenarios.

3.2 Out of Scope

The following topics are very important but are nonetheless out of the scope of this document:

- **Middleware.**

Middleware must be able to deal with situations that don't typically arise in end-user code, e.g., a variable number of arguments in a specific GridRPC call that is not known until call time.

- **Service Discovery.**

How the actual service registry or look-up is done is not addressed in this document. It is assumed that some type of registry or grid information service is available to accomplish this function.

- **Non-flat Service Names.**

The current API assumes simple name strings for GridRPC services. Describing and discovering GridRPC services by attributes or metadata schemas would certainly be very useful but is not addressed here.

- **General Workflow.**

Defining general mechanisms for managing grid workflows are not in the scope of this document. However, simple extensions to the API may be possible that allow the use of workflow management tools.

- **Interoperability between Implementations.**

Since this document focuses on the GridRPC API, it says nothing about the protocols used to communicate between clients, servers, and registries. Hence, it does not address interoperability.

4. The GridRPC API

We begin the presentation of the GridRPC API by defining the data types used. We then present the initialization/finalization calls, function handle management calls, the function calls themselves, and the control and wait calls. Each call definition includes a table of possible error codes that it can return.

4.1 GridRPC Data Types

grpc_function_handle_t

Variables of this data type represent a specific remote function that has been bound to a specific server which might be chosen by underlying GridRPC system. They are allocated by the user. After a *function handle* is initialized, it may be used to invoke the associated remote function as many times as desired. The lifetime of a *function handle* is determined when the user invalidates the *function handle* with a *handle destruct* call.

grpc_sessionid_t

Variables of this data type represent a specific non-blocking GridRPC call. *Session IDs* are used to probe or wait for call completion, to cancel a call, or to check the error status of a call. *Session IDs* are also allocated by the user but their lifetime is determined automatically. A *session ID* is initialized when a non-blocking GridRPC call is made. It is invalidated, or destroyed, when (1) all return arguments have been received, and (2) a wait function has returned a ``call complete'' status to the application. If an invalid *session ID* is passed to any GridRPC call, an error will result.

grpc_error_t

This data type is used for all error and return status codes from GridRPC functions.

4.2 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_error_t grpc_initialize(char *config_file_name)
This function reads the configuration file and initializes the required modules. After this function is called once, subsequent call will return with GRPC_ALREADY_INITIALIZED.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_CONFIGFILE_NOT_FOUND	Specified configuration file not found
GRPC_CONFIGFILE_ERROR	An error occurred parsing or processing the configuration file
GRPC_OTHER_ERROR_CODE	Internal error detected
GRPC_ALREADY_INITIALIZED	The function is called more than once.

grpc_error_t grpc_finalize(void)
This function releases any resources being used by GridRPC, canceling all the unfinished asynchronous calls.

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

4.3 Remote Function Handle Management Functions

The *function handle management* group of functions allows the creation and destruction of function handles.

grpc_error_t grpc_function_handle_default(
grpc_function_handle_t *handle,
char *func_name)
This creates a new function handle using a default server associated with the given function name. This default could be a pre-determined server or it could be a server that is dynamically chosen by the resource discovery mechanisms of the underlying GridRPC implementation. The server selection process could be postponed until the actual call is made on the handle. Once selection is made, all the calls through the handle must go to the server.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_SERVER_NOT_FOUND	GRPC client cannot find any server
GRPC_FUNCTION_NOT_FOUND	GRPC client cannot find the function on the default server
GRPC_RPC_REFUSED	Handle creation is refused by the server
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_function_handle_init(
grpc_function_handle_t *handle,
char *server_name,
char *func_name)

This creates a new function handle with a server explicitly specified by the user.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_SERVER_NOT_FOUND	GRPC client cannot find the specified server
GRPC_RPC_REFUSED	Handle creation is refused by the server
GRPC_FUNCTION_NOT_FOUND	GRPC client cannot find the function on the specified server
GRPC_OTHER_ERROR_CODE	Internal error detected

Advice to Implementors:
The exact form of the server name string is not specified. One common possibility is a string of the form ``*host_name:port_number*''. Another possibility is a string in some resource specification language.
End of Advice to Implementors.

grpc_error_t grpc_function_handle_destruct(grpc_function_handle_t *handle)

This releases all information and resources associated with the specified function handle. It also cancels a running session bound to the handle, if exists, before releasing the handle itself.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_INVALID_FUNCTION_HANDLE	Function handle pointed to by **handle** is not valid
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_get_handle(
grpc_function_handle_t **handle,
grpc_sessionid_t sessionID)

This returns the function handle corresponding to the given **session ID** (that is, corresponding to that particular non-blocking request).

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_INVALID_SESSION_ID	**sessionID** is not valid
GRPC_OTHER_ERROR_CODE	Internal error detected

4.4 GridRPC Call Functions

Two GridRPC call functions are available for end-users. These two calls are similar but provide either blocking (synchronous) or non-blocking (asynchronous) behavior. In the non-blocking case, a *session ID* is returned that is subsequently used to test for completion.

grpc_error_t grpc_call(grpc_function_handle_t *handle, <varargs>)
This makes a blocking remote procedure call with a variable number of arguments.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_SERVER_NOT_FOUND	GRPC client cannot find the specified server
GRPC_FUNCTION_NOT_FOUND	GRPC client cannot find the function on the specified server
GRPC_INVALID_FUNCTION_HANDLE	Function handle pointed to by **handle** is not valid
GRPC_RPC_REFUSED	RPC invocation refused by the server, possibly because of a security issue
GPRC_COMMUNICATION_FAILED	Communication with the server failed somehow
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_call_async(
grpc_function_handle_t *handle,
grpc_sessionid_t *sessionID,
<varargs>)
This makes a non-blocking remote procedure call with a variable number of arguments. 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.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_SERVER_NOT_FOUND	GRPC client cannot find the specified server
GRPC_FUNCTION_NOT_FOUND	GRPC client cannot find the function on the specified server
GRPC_INVALID_FUNCTION_HANDLE	Function handle pointed to by **handle** is not valid
GRPC_RPC_REFUSED	RPC invocation refused by the server, possibly because of a security issue
GPRC_COMMUNICATION_FAILED	Communication with the server failed somehow
GRPC_OTHER_ERROR_CODE	Internal error detected

The GridRPC Recommendation 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.

Rationale:

By returning as soon as possible, e.g., before the remote operation has started and before any results are returned, the GridRPC user can overlap the remote computation with other local computation. By allowing the user to modify any buffers after the asynchronous call returns, we present the user with the safest and simpliest buffer handling semantics possible. While it may be possible to improve performance further by allowing asynchronous calls to return before it is safe to modify input argument buffers, it was considered not worth the added complexity and ``danger'' for handling buffers. This is the approach taken by current prototype implementations. *End of Rationale.*

4.5 Asynchronous GridRPC Control Functions

The following functions apply only to previously submitted non-blocking requests.

grpc_error_t grpc_probe(grpc_sessionid_t sessionID)
This call checks whether the asynchronous GridRPC call represented by the *session ID* **sessionID** has completed. If it has completed, **GRPC_NO_ERROR** is returned. Otherwise, **GRPC_NOT_COMPLETED** is returned.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_INVALID_SESSION_ID	**sessionID** is not valid
GPRC_NOT_COMPLETED	Call has not completed
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_probe_or(
grpc_sessionid_t *idArray,
size_t length,
grpc_sessionid_t *idPtr)
This call checks whether the asynchronous GridRPC calls represented by the array of *session IDs* in **idArray** have completed. If any calls have completed, the function return value is **GRPC_NO_ERROR** and the **grpc_sessionid_t** pointed to by ***idPtr** contains exactly one valid, completed call. If no call has completed, the function return value is **GRPC_NONE_COMPLETED** and the **grpc_sessionid_t** pointed to by ***idPtr** is undefined. If any of the *session IDs* in **idArray** are invalid, no operations will occur and an **GRPC_INVALID_SESSION_ID** error will be returned. However, the array of *session IDs* may contain completed *session IDs* without causing an error.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_INVALID_SESSION_ID	A *session ID* in **idArray** is not valid
GPRC_NONE_COMPLETED	No calls in **idArray** have completed
GRPC_OTHER_ERROR_CODE	Internal error detected

Rationale:
Users will typically fill a such an array with *session IDs* and then check for them to finish one by one. Hence, it will be a common occurrence that such an array may contain completed *session*

IDs. If having a sparse array presents a performance concern, the user has the option of packing the array themselves.
End of Rationale.

Advice to Implementors:
While this document does not specify the actual representation for **grpc_sessionid_t**, it would be possible for implementations to use some value to denote *void* or *invalid* variables of this type that could be used for internal error-checking. For example, when **grpc_probe_or()** returns **GRPC_NONE_COMPLETED**, the **grpc_sessionid_t** pointed to by **idPtr** could actually be set to this *void* value.
End of Advice to Implementors.

grpc_error_t grpc_cancel(grpc_sessionid_t sessionID)
This cancels the specified asynchronous GridRPC call.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_INVALID_SESSION_ID	**sessionID** is not valid
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_cancel_all(void)
This cancels all outstanding asynchronous GridRPC calls.

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

Rationale:
A ``cancel array'' call was considered but dismissed since it would cause difficult error handling.
End of Rationale.

4.6 Synchronous GridRPC Wait Functions
The following five functions apply only to previously submitted non-blocking requests. These calls allow an application to express desired non-deterministic completion semantics to the underlying system, rather than repeatedly polling on a set of *session IDs*.

Advice to Implementors:
From an implementation standpoint, such information could be conveyed to the OS scheduler to reduce cycles wasted on polling.
End of Advice to Implementors.

grpc_error_t grpc_wait(grpc_sessionid_t sessionID)
This blocks until the specified non-blocking requests to complete.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet

GRPC_INVALID_SESSION_ID	**sessionID** is not valid
GPRC_COMMUNICATION_FAILED	Communication with the server failed somehow
GRPC_SESSION_FAILED	The specified session failed
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_wait_and(
grpc_sessionid_t *idArray,
size_t length)
This blocks until *all* of the specified non-blocking requests in **idArray** have completed.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_INVALID_SESSION_ID	One or more *session IDs* in **idArray** are not valid
GRPC_SESSION_FAILED	One or more sessions failed
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_wait_or(
grpc_sessionid_t *idArray,
size_t length,
grpc_sessionid_t *idPtr)
This blocks until *any* of the specified non-blocking requests in **idArray** has completed. On a successful return, **idPtr** points to a completed request.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_INVALID_SESSION_ID	One or more *session IDs* in **idArray** are not valid
GRPC_SESSION_FAILED	The *session ID* pointed to by idPtr failed
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_wait_all(void)
This blocks until *all* previously issued non-blocking requests have completed.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_SESSION_FAILED	One or more sessions failed
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_wait_any(grpc_sessionid_t *idPtr)

This blocks until *any* previously issued non-blocking requests has completed. On a successful return, **idPtr** points to a completed request.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_SESSION_FAILED	The *session ID* pointed to by **idPtr** failed
GRPC_OTHER_ERROR_CODE	Internal error detected

Note: For **grpc_wait_or()** and **grpc_wait_any()**, if more than one call has completed, it is undefined which completed *session ID* is pointed to by **idPtr**. That is to say, if more than one call has completed, it is not guaranteed that the *session ID* returned is for the call that actually completed first, or that successive *wait* calls will return completed *session IDs* in any particular order.

4.7 Error codes and Error Reporting Functions

When a GridRPC call fails, an error code is returned. Table 1 gives the error code identifiers that can be used with variables of type *grpc_error_t*.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_CONFIGFILE_NOT_FOUND	Specified configuration file not found
GRPC_CONFIGFILE_ERROR	An error occurred parsing or processing the configuration file
GRPC_SERVER_NOT_FOUND	GRPC client cannot find any server
GRPC_FUNCTION_NOT_FOUND	GRPC client cannot find the function on the default server
GRPC_INVALID_FUNCTION_HANDLE	Function handle is not valid
GRPC_INVALID_SESSION_ID	*Session ID* is not valid
GRPC_RPC_REFUSED	RPC invocation refused by the server, possibly because of a security issue
GRPC_COMMUNICATION_FAILED	Communication with the server failed somehow
GRPC_SESSION_FAILED	The specified session failed
GRPC_NOT_COMPLETED	Call has not completed
GRPC_NONE_COMPLETED	No calls have not completed
GRPC_OTHER_ERROR_CODE	Internal error detected
GRPC_UNKNOWN_ERROR_CODE	Error description string requested for an unknown error code
GRPC_ALREADY_INITIALIZED	grpc_initialize is called more than once.
GRPC_LAST_ERROR_CODE	Highest numerical error code; used to bound error codes and does not denote an actual error

Table 1. Summary of GridRPC Error Codes

These error codes satisfy:

0 = GRPC_NO_ERROR < GRPC_... < GRPC_LAST_ERROR_CODE

This specifies a useful numerical ordering of the error codes based on the set of integers without specifying a specific implementation.

The ability to check the error code of previously submitted requests is provided. The following error reporting functions provide error codes and human-readable error descriptions. These error descriptions can be more informative about the actual cause of the error.

char *grpc_error_string(grpc_error_t error_code)
This returns the error description string, given a GridRPC error code. If the error code is unrecognized for any reason, the string **GRPC_UNKNOWN_ERROR_CODE** is returned.

grpc_error_t grpc_get_error(grpc_sessionid_t sessionID)
This returns the error code associated with a given non-blocking request.

Error Code Identifier	Meaning
GRPC_NO_ERROR	Success
GRPC_NOT_INITIALIZED	GRPC client not initialized yet
GRPC_INVALID_SESSION_ID	**sessionID** is not valid
GPRC_COMMUNICATION_FAILED	Communication with the server failed somehow
GRPC_SESSION_FAILED	The specified session failed
GRPC_OTHER_ERROR_CODE	Internal error detected

grpc_error_t grpc_get_failed_sessionid(grpc_sessionid_t *idPtr)
This returns the *session ID* associated with the most recent **GRPC_SESSION_FAILED** error. This provides additional error information on a specific *session ID* that failed for calls that deal with sets of *session IDs*, either implicitly, such as **grpc_wait_all()**, or explicitly, such as **grpc_wait_and()**. When there are more than two failed sessions, this function will return the session ID one by one. To make sure that all the failed sessions are handled, users have to call this function repeatedly until it returns GRPC_SESSIONID_VOID.

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

Rationale:

The GridRPC error codes are intended to be similar to error classes in the MPI standard. That is to say, these are types of errors that are inherent to the GridRPC API and may occur in any GridRPC implementation. Implementation-specific error information may be contained in the associated error description strings. The **GRPC_OTHER_ERROR_CODE** error code may be used for implementation-specific errors.

End of Rationale.

5. Related Work

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, 20], and Ninf [14, 19]. 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 (http://punch.purdue.edu). 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 (http://www.xml-rpc.com) and SOAP [18] 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 [17] has shown that GridRPC could be effectively built upon future Grid software based on Web Services such as OGSA [8].

6. Security Considerations

Security issues of GridRPC are implementation-dependent and this document does not specifically address security in the API. For reference, security mechanisms of Ninf-G and NetSolve are described in this section. Security infrastructure of Ninf-G is based on GSI which is based on public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol. This means that not only all the components are protected properly, but they can also utilize other Globus components, such as GridFTP servers, seamlessly and securely. NetSolve's current security is based on the ability to generate access control lists that are used to grant and deny access to the NetSolve servers. NetSolve uses Kerberos V5 services for authentication. The Kerberos extensions of NetSolve provide it with trusted mechanisms by which to control access to computational resources. At this time, the Kerberized version of NetSolve performs no encryption of the data exchanged among NetSolve clients, servers, or agents, nor is there any integrity protection for the data stream.

Author Contact Information

Hidemoto Nakada
National Institute 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
University of Tennessee, Knoxville
Seymour@cs.utk.edu

Jack Dongarra
University of Tennessee, Knoville
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 Supercomputer Center
Casanova@cs.ucsd.edu

Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to 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. https://forge.gridforum.org/projects/gridrpc-wg/
[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, M. Sato, and S. Sekiguchi. Design and Implementation of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15(5-6):649-658, 1999.
[15] K. Seymour et al. An Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In *Proceedings of the 3rd International Workshop on Grid Computing, volume 2536, pages 274-278. Springer-Verlag, Lecture Notes in Computer Science*, November 2002.
[16] K. Seymour et al. GridRPC: A Remote Procedure Call API for Grid Computing. In *Proceedings of the International Workshop on Grid 2002*, pages 274-278, 2002.
[17] S. Shirasuna, H. Nakada, S. Matsuoka, and S. Sekiguchi. Evaluating Web Services Based Implementation of GridRPC. In *Proceedings of HPDC11*, pages 237-245, 2002.
[18] Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/SOAP, May 2000, W3C Note.
[19] Y. Tanaka, H. Nakada, S. Sekiguchi, T. 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.
[20] D. Arnold, S. Browne, J. Dongarra, G. Fagg, and K. Moore. Secure Remote Access to Numerical Software and Computational Hardware. In *Proceedings of the DoD HPC Users Group Conference (HPCUG) 2000*, 2000.
[21] T. Suzumura, T. Nakagawa, S. Matsuoka, H. Nakada, and S. Sekiguchi, Are Global Computing Systems Useful? - Comparison of Client-Server Global Computing Systems Ninf,

NetSolve versus CORBA, In Proceedings of International Parallel and Distributed Processing Symposium 2000, 2000.