Interoperability Testing for The GridRPC API Specification

Status of This Memo

This memo provides information regarding the testing of interoperability for implementations of the GridRPC API specification. Distribution is unlimited.

Abstract

This document proposes a process of interoperability testing of the GridRPC API specification, which is submitted to the Global Grid Forum (GGF) recommendation track. Three independent implementations from different code bases, Ninf-G, GridSolve and DIET, are used for showing sufficient successful operational experiences. A strong belief that the specification is mature and will be useful is indicated through the test cases.

Contents

1. Introduction

The initial GridRPC API specifications of the GridRPC Working Group have been submitted to the GGF Recommendation process [GFD-R.52]. This document describes a process by which it is proposed to test the specifications for interoperability, as required in [GFD.1]. In this document, the interoperable API means to be functionally equivalent on each implementation that provides the API. In the case of the GridRPC API, the client program code should be shared between each implementation, and should behave the same.

The document is specific to the GridRPC API specifications in two respects:

1. The nature of the interoperability required. Implementations of the specifications do not call themselves or each other. As such, the requirement is only to test client interoperability: that multiple implementations of a specification provide consistent results to a suite of tests.
2. The challenges identified for designers of a comprehensive test regime, which follow from certain features of the specifications.

The GridRPC API specifications are implemented in three different code-based systems, Ninf-G [Ninf-G], GridSolve [GridSolve] and DIET [DIET], which are applied to the interoperability test.

1.1 Ninf-G

Ninf-G (http://ninf.apgrid.org/) provides the GridRPC API on top of the Globus Toolkit [Globus]. 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. Ninf-G is maintained by the Ninf project, which consists of National Institute of Advanced Industrial Science and Technology (AIST), and several universities in Japan.

1.2 GridSolve

GridSolve (http://icl.cs.utk.edu/gridsolve/) is a client-server system which provides remote access to computational resource, both hardware and software. It is built upon standard Internet protocols, like TCP/IP sockets. GridSolve provides the GridRPC API as C interface, and another API as Matlab interface. GridSolve is maintained by University of Tennessee in the United States.

1.3 DIET

DIET (http://graal.ens-lyon.fr/DIET) is a toolbox to develop a set of tools to build computational servers and clients. Based on client-server system which provides remote access to computational resource, both hardware and software as GridSolve. It is built upon CORBA communication protocols. DIET provides the GridRPC API as C interface. DIET is maintained by the GRAAL INRIA project in École Normale Supérieure Lyon in France.

2. Proposed Process

The aim of the process is to test for client interoperability. A set of test programs that use the GridRPC APIs is tried to be compiled with each GridRPC library, and executed on each GridRPC system. Note it is out of the specification that the test program linked to a particular client library communicates with different server side implementations. None of the GridRPC implementations support this feature.

1. Define test cases that can be used as the basis of interoperability testing. These cases will include checks of the API function and its error code, in non-error and error situation.
2. Implement the test cases.
3. Compile the test programs, checking the program in client-side linked with each GridRPC library without any modification.
4. Execute the test programs, checking the test passed in each case.
5. Document the test cases and the results.

The above test cases have been prepared so that they cover all functions of the API in the specification. Until that the all test cases succeed on a GridRPC implementation, the implementation is not "GridRPC compliant." The test cases, however, are not intended to check whether the implementation is "GridRPC compliant" or not. The result that the all test cases are successful does not promise that the implementation is "GridRPC compliant."

3. Testing

The Ninf-G version 4.1.0 that was released on April 10, 2006, and the GridSolve version 0.15 that was released on May 8, 2006, and DIET version 2.1 (public version coming soon in September 2006) were used for this interoperability test.

3.1 Initializing and Finalizing Functions

The following functions for initialization and finalization of the GridRPC system are tested.

```
grpc_error_t grpc_initialize(char *config_file_name)
grpc_error_t grpc_finalize(void)
```

3.1.1 Notes for Results

The configuration file is independent from the GridRPC API specification. The error codes, GRPC_CONFIGFILE_NOT_FOUND and GRPC_CONFIGFILE_ERROR, are optional. Ninf-G and DIET require a configuration file but GridSolve does not it.

3.1.2 Test Cases

1. Call grpc_initialize() with a correct configuration file, checking GRPC_NO_ERROR returned.
2. Call grpc_initialize() with a correct configuration file twice, checking GRPC_ALREADY_INITIALIZED returned.
3. Call grpc_finalize() in right way, checking GRPC_NO_ERROR returned.
4. Call grpc_finalize() before calling gprc_initialize(), checking GRPC_NOT_INITIALIZED returned.

3.2 Remote Function Handle Management Functions

The following functions for creating and destructing the function handle are tested.

```
grpc_error_t grpc_function_handle_default(
        grpc_function_handle_t *handle,
        char *func_name)
grpc_error_t grpc_function_handle_init(
        grpc_function_handle_t *handle,
        char *server_name,
        char *func_name)
grpc_error_t grpc_function_handle_destruct(grpc_function_handle_t *handle)
grpc_error_t grpc_get_handle(
        grpc_function_handle_t **handle,
        grpc_sessionid_t sessionID)
```

3.2.1 Notes for Results

The error codes, GRPC_FUNCTION_NOT_FOUND and GRPC_SERVER_NOT_FOUND, are optional. Ninf-G and DIET return those error codes in the remote function handle management functions, and GridSolve returns them in the GridRPC call functions.

3.2.2 Test Cases

1. Call grpc_function_handle_default() with an available function name, checking GRPC_NO_ERROR returned with a pointer of the initialized function handle.
2. Call grpc_function_handle_default() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.
3. Call grpc_function_handle_init() with an available set of the function name and the server name, checking GRPC_NO_ERROR returned with a pointer of the initialized function handle.
4. Call grpc_function_handle_init() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.
5. Call grpc_function_handle_destruct() in right way, checking GRPC_NO_ERROR returned.
6. Call grpc_function_handle_destruct() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.
7. Call grpc_get_handle() with a valid session ID, checking GRPC_NO_ERROR returned with a pointer of the function handle specified by ID.
8. Call grpc_get_handle() with an invalid session ID, checking GRPC_INVALID_SESSION_ID returned.
9. Call grpc_get_handle() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.

3.3 GridRPC Call Functions

The following functions for invoking the RPC are tested.

```
grpc_error_t grpc_call(
        grpc_function_handle_t *handle,
        <varargs>)
grpc_error_t grpc_call_async(
        grpc_function_handle_t *handle,
        grpc_sessionid_t *sessionID,
        <varargs>)
```

3.3.1 Notes for Results

The error codes, GRPC_FUNCTION_NOT_FOUND and GRPC_SERVER_NOT_FOUND, are optional. Ninf-G and DIET return those error codes in the remote function handle management functions, and GridSolve returns them in the GridRPC call functions. Moreover, it's not usual but if the service disappears between the remote function handle and the GridRPC call functions, DIET returns a GRPC_FUNCTION_NOT_FOUND from the call function too.

3.3.2 Test Cases

1. Call grpc_call() with an initialized handle and valid arguments, checking GRPC_NO_ERROR returned with correct output arguments.
2. Call grpc_call() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.
3. Call grpc_call_async() with an initialized handle and valid arguments, checking GRPC_NO_ERROR returned with a valid session ID.
4. Call grpc_call_async() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.

3.4 Wait Functions

The following function for waiting for previously submitted non-blocking requests.

```
grpc_error_t grpc_wait(grpc_sessionid_t sessionID)
grpc_error_t grpc_wait_and(
        grpc_sessionid_t *idArray,
        size_t length)
grpc_error_t grpc_wait_or(
        grpc_sessionid_t *idArray,
        size_t length,
        grpc_sessionid_t *idPtr)
grpc_error_t grpc_wait_all(void)
grpc_error_t grpc_wait_any(grpc_sessionid_t *idPtr)
```

3.4.1 Notes for Results

There is no difference for the test results among each GridRPC implementation.

3.4.2 Test Cases

1. Call grpc_wait() with a valid session ID to be synchronized, checking GRPC_NO_ERROR returned.
2. Call grpc_wait() with an invalid session ID, checking GRPC_INVALID_SESSION_ID returned.
3. Call grpc_wait() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED retuned.
4. Call grpc_wait_and() with an array of the valid session IDs, checking GRPC_NO_ERROR returned.
5. Call grpc_wait_and() with a session ID array which contains at least one invalid ID, checking GRPC_INVALID_SESSION_ID returned.
6. Call grpc_wait_and() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED retuned.
7. Call grpc_wait_or() with a valid session ID array, checking GRPC_NO_ERROR returned with a pointer of the completed session ID.
8. Call grpc_wait_or() with a session ID array which contains at least one invalid ID, checking GRPC_INVALID_SESSION_ID returned.

9. Call grpc_wait_or() before calling grpc_wait_or(), checking GRPC_NOT_INITIALIZED returned.
10. Call grpc_wait_all() in right way, checking GRPC_NO_ERROR returned.
11. Call grpc_wait_all() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.
12. Call grpc_wait_any() in right way, checking GRPC_NO_ERROR retuned with an ID pointer of the session which is completed.
13. Call grpc_wait_any() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.

3.5 Probe Functions

The following functions for checking whether previously submitted non-blocking requests are completed are tested.

```
grpc_error_t grpc_probe(grpc_sessionid_t sessionID)
grpc_error_t grpc_probe_or(
        grpc_sessionid_t *idArray,
        size_t length,
        grpc_sessionid_t *idPtr)
```

3.5.1 Notes for Results

There is no difference for the test results among each GridRPC implementation.

3.5.2 Test Cases

1. Call grpc_probe() with a session ID which is completed, checking GRPC_NO_ERROR returned.
2. Call grpc_probe() with a session ID which is not completed, checking GRPC_NOT_COMPLETED returned.
3. Call grpc_probe() with an invalid session ID, checking GRPC_INVALID_SESSION_ID returned.
4. Call grpc_probe() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.
5. Call grpc_probe_or() with a session ID array which contains at least one completed session, checking GRPC_NO_ERROR returned with a pointer of the probed session ID.
6. Call grpc_probe_or() with a session ID array which does not contain any of completed sessions, checking GRPC_NONE_COMPLETED returned with an invalid session ID.
7. Call grpc_probe_or() with a session ID array which contains an invalid session ID, checking GRPC_INVALID_SESSION_ID returned.
8. Call grpc_probe_or() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.

3.6 Cancel Functions

The following functions for canceling previously submitted non-blocking requests are tested.

```
grpc_error_t grpc_cancel(grpc_sessionid_t sessionID)
grpc_error_t grpc_cancel_all(void)
```

3.6.1 Notes for Results

There is no difference for the test results among each GridRPC implementation.

3.6.2 Test Cases

1. Call grpc_cancel() with a valid session ID, checking GRPC_NO_ERROR returned after the specified session is canceled.
2. Call grpc_cancel() with an invalid session ID, checking GRPC_INVALID_SESSION_ID returned.
3. Call grpc_cancel() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.
4. Call grpc_cancel_all() in right way, checking GRPC_NO_ERROR returned after all of the executing sessions are canceled.
5. Call grpc_cancel_all() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.

3.7 Error Reporting Functions

The following functions for reporting an error are tested.

```
char* grpc_error_string(grpc_error_t error_code)
grpc_error_t grpc_get_error(grpc_sessionid_t sessionID)
grpc_error_t grpc_get_failed_sessionid(grpc_sessionid_t *idPtr)
```

3.7.1 Notes for Results

The error code, GRPC_NOT_INITIALIZED, is not defined in the error reporting functions. When the functions are called before grpc_initialize() is called, however, some GridRPC implementations might return GRPC_NOT_INITIALIZED.

3.7.2 Test Cases

1. Call grpc_error_string() with a defined error code, checking the corresponded error string returned.
2. Call grpc_error_string() with a non-defined error code, checking the string which means “GRPC_UNKNOWN_ERROR CODE” returned.
3. Call grpc_get_error() with a valid session ID, checking the error code associated with the given session returned.
4. Call grpc_get_error() with an invalid session ID, checking GRPC_INVALID_SESSION_ID returned.
5. Call grpc_get_failed_sessionid() when there is no failed session, checking GRPC_NO_ERROR returned.
6. Call grpc_get_failed_sessionid() when there is one failed session, checking a pointer of that session ID returned as idPtr.
7. Call grpc_get_failed_sessionid() repeatedly when there are more than two failed sessions, checking a pointer of that session ID returned one by one, until all of them are popped out.

4. Security Considerations

There is no notion on security consideration in the specification. Each implementation has a different security model underneath the GridRPC API. For instance, 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.

No security mechanism is implemented in the GridSolve version 0.15. There is a pre-existing system of GridSolve, which is called NetSolve. The security of the NetSolve version 2.0 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.

No security mechanism is implemented in DIET 2.x at this time. In the past, users using DIET and who require security have used a network solution (VPN). Another solution can be built on security policies available into CORBA ORB implementation. When the security designed in CORBA specification will be available in omniORB (CORBA ORB) then DIET will use this security.

5. Conclusion

There are several minor differences among Ninf-G, GridSolve and DIET. The differences come from the basic design of each implementation. When users start to use one of implementations or move from another implementation to the other, they usually learn features of each GridRPC system. The differences are not beyond of this kind of learning but understandable. On the other hand, three behaviors are not defined in the specifications. The detailed information is described in Appendix A. Except those differences, all test cases passed in Ninf-G, GridSolve and DIET, which means the interoperability of the client code was confirmed. Therefore, the GridRPC API specification (GFD-52) will be useful for people, who would like to write their program on the GridRPC framework or develop another GridRPC system.

Author Information

Yusuke Tanimura,
National Institute of Advanced Industrial Science and Technology (AIST),
1-1-1 Umezono, Tsukuba Central 2,
Tsukuba, Ibaraki,
Japan.

Keith Seymour,
University of Tennessee,
1122 Volunteer Blvd, Suite 413 Claxton,
Knoxville, Tennessee,
USA.

Eddy Caron
ENS-Lyon / LIP / INRIA
46 allée d'Italie
69364 Lyon
France

Abdelkader Amar
ENS-Lyon / LIP / INRIA
46 allée d'Italie

69364 Lyon
France

Hidemoto Nakada,
National Institute of Advanced Industrial Science and Technology (AIST),
1-1-1 Umezono, Tsukuba Central 2,
Tsukuba, Ibaraki,
Japan.

Yoshio Tanaka,
National Institute of Advanced Industrial Science and Technology (AIST),
1-1-1 Umezono, Tsukuba Central 2,
Tsukuba, Ibaraki,
Japan.

Frédéric Desprez
ENS-Lyon / LIP / INRIA
46 allée d'Italie
69364 Lyon
France

Intellectual Property Statement

Full Copyright Notice

References

[GFD.1]
C. Catlett, GGF Document Series, Global Grid Forum Document GFD.1 (http://www.gridforum.org/documents/GFD.1.pdf).

[GFD-R.52]
H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova, A GridRPC Model and API for End-User Applications, Global Grid Forum Document GFD-R.52.

[GridRPC]
Keith Seymour, H.Nakada, S. Matsuoka, Jack Dongarra, Craig Lee, Henri Casanova, "Overview of GridRPC: A Remote Procedure Call API for Grid Computing", (GRID COMPUTING - GRID 2002, LNCS 2536), pp274-278, November, 2002.

[Ninf-G]
Y.Tanaka, H.Takemiya, H.Nakada, S.Sekiguchi, "Design, Implementation and Performance Evaluation of GridRPC Programming Middleware for a Large-scale Computational Grid", International Workshop on Grid Computing, November, 2004.

[GridSolve]
S. Agrawal, J. Dongarra, K. Sagi, K. Seymour, A. YarKhan, “Users’ Guide to GridSolve Version 0.14”, ICL, University of Tennessee, March, 2006.

[DIET]
E. Caron and F. Desprez, “DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid”, International Journal of High Performance Computing Applications, 2006.

[Globus]
I. Foster and C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit”, International Journal of Supercomputer Applications, 1997.

Appendix A: Undefined behaviors

This section mentions about undefined behaviors though they are not concerned in the interoperability test and not requirements for being GridRPC compliant. While these undefined behaviors are a minor problem, they should be clearly defined in the future version of the specification according to the use cases. In the current specification, the following information would be helpful for users because they might see these differences in their development with each GridRPC implementation.

A.1 Lifetime of the session ID

In the specification document, lifetime of the session ID is described below.

> *"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."*

Because lifetime of the failed session ID is not defined, the lifetime is handled differently among each GridRPC implementation. In Ninf-G, the session ID will be valid for a certain period if the save_sessionInfo parameter is set to more than 0. The value of save_sessionInfo indicates a rotation of the session IDs. If the session ID is valid, the probe functions and the error reporting functions will return the session information after the session is completed or canceled. The wait functions will return the specific error code after the session is canceled. If save_sessionInfo is set to 0, the session ID is invalid after the session is completed, canceled or aborted by the error.

In GridSolve, the session ID becomes invalid immediately after the session is completed at the wait functions or canceled at the cancel functions. The probe functions, the error reporting functions and the wait functions will just return GRPC_INVALID_SESSION_ID after the session is completed or canceled. On the other hand, the failed session ID is saved in the library. The error reporting function returns the specific error code after the session is aborted by the error.

In DIET, the session ID becomes invalid immediately after the session is canceled at the cancel functions or after the grpc_finalize() function. The probe functions, the error reporting functions and the wait functions will just return GRPC_INVALID_SESSION_ID after the session is canceled. If a session fails, the session ID will be still valid and the error code can be retrieved by grpc_get_error() with the session ID.

Certainly, this undefinition should be resolved in the future version of the specification according to the use cases. The issue may be not only related to the error session. The session information might be useful for statistics of the GridRPC calls in the application layer or might not be necessary for applications.

A.2 Destruction of the function handle

Destruction procedure of the function handle that has a running session is not strictly defined in the specification. The related description in the specification document is below.

"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."

The undefined behavior is whether the GridRPC implementation needs to confirm cleanup on the server side is completed or not. The Ninf-G users can specify the job_stopTimeout variable in the client configuration file. If the value is positive, the timeout will occur after the given seconds. In addition, Ninf-G will show a warning message if the function handle to be destructed has a running session. GridSolve and DIET do not confirm completeness of cleanup on the server side and no warning message is shown to users.

There is a similar issue in grpc_finalize(). In Ninf-G and DIET, grpc_finalize() invokes a cancellation request to the server but it returns immediately without waiting for completeness of any cancellation. In GridSolve, grpc_finalize() does not send a cancellation request to the server explicitly.

A.3 Initialization of the function handle

When an invalid hostname is specified in the argument of grpc_function_handle_init(), different error codes will be returned. In Ninf-G, the error code is always GRPC_OTHER_ERROR_CODE. In GridSolve, GRPC_OTHER_ERROR_CODE will be returned if the hostname is not resolved. If the hostname is resolved, the error code will be GRPC_NO_ERROR and another error such as GRPC_RPC_REFUSED will be returned at the GridRPC call functions. In DIET the hostname is fake information, because the aim of the DIET scheduler is to find the best server and this server is linked to the GridRPC hostname.

Appendix B: Test code

This interoperability test code is available online at https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.gridrpc-wg/docman.root.interoperabilitytest/doc8591/1.