Interoperability Testing for The GridRPC API Specification

Y. Tanimura, Keith Seymour, Eddy Caron, Abdelkader Amar, Hidemoto Nakada, Yoshio Tanaka, Frédéric Desprez

GFD-E.102
GridRPC Working Group

Y. Tanimura, AIST
K. Seymour, Univ. of Tenn.
E. Caron, INRIA
A. Amar, INRIA
H. Nakada, AIST
Y. Tanaka, AIST
F. Desprez, INRIA

August 11, 2006
Updated: April 5, 2007

Interoperability Testing for The GridRPC API Specification

Status of This Document

This document provides information to the Grid community regarding the testing of interoperability for implementation of the GridRPC API specification. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

Abstract

This document proposes a process of interoperability testing of the GridRPC API specification (GFD-R.52), 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 all the results of the test suites.

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 general, "interoperable" means to be functionally equivalent or interchangeable components of the system or process in which they are used. However, there is a following statement in Section 3.2 of GFD-R.52.

> ***- 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.*

TheGFD-R.52 defines a GridRPC model and client APIs. The defining APIs is intended to share a GridRPC client program between each GridRPC implementations, and it is tremendous useful for application users. In this document, therefore, the purpose of the interoperability test is to prove the specification is mature and useful enough to share the client program among GridRPC-compliant systems. In particular, if the client APIs is implemented correctly on each GridRPC system, according to the specification, and if the specification does not leave ambiguity or impossibility for middleware developers are confirmed. The interoperability test covers all functions of the GridRPC API specification, meaning that the test is equal to the conformance test of the GridRPC API.

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 test suites.

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, executed on each GridRPC system, and checked if it behaved the same. Note it is out of the specification that the test program linked to a particular client library communicates with different server side implementations but it is unrealistic that a GridRPC implementation does not support the 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 (test suites), 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. In other words, again, this is a conformance test of the GridRPC End-User API. When the all test cases succeeded on a GridRPC implementation, the implementation is "GridRPC (GFD-R.52) compliant."

3. Testing

The Ninf-G version 4.2.1 that was released on November 20, 2006, and the GridSolve version 0.15.1 that was released on September 29, 2006, and DIET version 2.3 that is planned to be released in June 2007 were used for this interoperability test. Test suites shown in Appendix B were used for this test. Because GFD-R.52 only defines C language API, the test was only applied to C bindings of each implementation.

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

There is no notion about necessity of the configuration file and also inside of the file, in Section 4.2 of GFD-R.52. 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 because the GFD-R.52 defines both error codes in the remote function handle management functions (in Section 4.3 of GFD-R.52) and in the GridRPC call functions (in Section 4.4 of GFD-R.52). 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, in Section 4.7 of GFD-R.52. As a result, when the functions were called before grpc_initialize() was called, some GridRPC implementations did not return GRPC_NOT_INITIALIZED. The problem was fixed in the implementations because this is a bug of the specification. The error codes of grpc_get_error() and grpc_get_failed_sessionid() are going to be defined in the final version of GFD-R.52.

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_error() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.
6. Call grpc_get_failed_sessionid() when there is no failed session, checking GRPC_NO_ERROR returned.
7. Call grpc_get_failed_sessionid() when there is one failed session, checking a pointer of that session ID returned as idPtr.

8. 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.
9. Call grpc_get_failed_sessionid() before calling grpc_initialize(), checking GRPC_NOT_INITIALIZED returned.

4. Security Considerations

There is no notion on security consideration, which is explicitly described in Section 6 of GFD-R.52. 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 straightforwardly understood during the learning process. In addition, three behaviors are intentionally left unspecified in the specification. The detailed information is described in Appendix A. All of the undefinitions are not critical and should be defined in the future, on the basis of more use cases. The undefinition explained in Section 3.7 is a bug of the specification and will be solved in the final version of GFD-R.52. Except those differences and undefinitions, 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 a GridRPC system.

6. Contributors

This document is the result of the joint efforts of many contributors. The Authors listed on the title page are those committed to taking permanent stewardship for this document – receiving communication in the future and otherwise being responsive to its content.

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.

In addition, the authors would like to thank all contributors from OGF's GridRPC working group attendees and developers of Ninf-G, GridSolve and DIET.

7. Intellectual Property Statement

8. Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

9. Full Copyright Notice

10. 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 describes undefined behaviors found through the interoperability test. All the undefined behaviors are implementation dependent and not included in the requirements for being GridRPC compliant. Rather, they should not be strictly defined in the current specification until use cases enough for defining them will be obtained. They leave the GridRPC system developers free to implement and should be clearly defined in the future version of the specification. The following information is provided for application developers and users to show possible differences between GridRPC implementations.

A.1 Lifetime of the session ID

In Section 4.1 of the GFD-R.52 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 is expected to 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 Section 4.3 of the GFD-R.52 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

In Section 4.3 of GFD-R.52, there is a following advice to implementors for initialization of the function handle.

> *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.*

All the three implementations accept hostname as server_name argument. 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 suites

This interoperability test suites are available online at https://forge.gridforum.org/sf/go/doc14379?nav=1.