Proposal for a data management API in Grid RPC

GRAAL Research Team

March 5, 2004

1 Introduction

This document takes part in the work on standardizing the Grid RPC remote procedure call interface for Grid computing. This paper follows the presentation first given at the Global Grid Forum 8 (Seattle) on data management in Grid RPC. It proposes to add a data handle as well as a set of functions for data management to the general Grid RPC interface. These functions will allow a NES using the GridRPC platform to provide data persistence at the user level. Data will be able to be kept on the platform between different requests client. If data dependences exist between requests, useless data transfers will be avoided.

The work presented here has already been validated by an implementation in the DIET [1] platform. The prototype manages data across the whole platform transparently to the client. It does not need to manage its data mapping on the platform. To achieve such a goal, we provide some functionalities to the programmer. Our mechanisms are based on two key points: a data must be fully identified by a data handle and a programmer must be able to choose whether a data has to be persistent inside the platform or not. Of course, the last point can be automatically generated by a high level scheduling tool if the whole workflow graph is available.

The first section presents the data handle object that will identify a data in the platform. The second section describes the functions used by the client to remotely manage its data.

2 Data handle

To identify a data item mapped on the platform, a *Data Handle* is needed. It has to be created the first time the data is used on the Grid platform. Then,

[1] `http://graal.ens-lyon.fr/DIET/`

it will be used to identify the data item for each request and to manage the data persistence on the platform.

The generic type for a data handle is `grpc_data_handle_t`. It may be implemented, for instance, as a structure recording the client Session ID and a data identifier, if the data is managed directly by the platform, or by a file descriptor, if the data management is provided by a file storage server. One important information is the persistency flag described further.

2.1 Managing Data Handles

The data handle management is based on the following functions:

```
grpc_new_data_handle(grpc_data_handle_t *data_handle,
                     <varargs>)
```

is used to declare a new data handle. It takes a pointer to a `grpc_data_handle_t` as an argument and returns the handle created.

```
grpc_free_data_handle(grpc_data_handle_t *data_handle)
```

is used to remove a data handle from the platform. It takes a pointer to a `grpc_data_handle_t` as an argument.

```
grpc_read_data_handle(grpc_data_handle_t *data_handle,
                <varargs>)
grpc_write_data_handle(grpc_data_handle_t *data_handle,
                <varargs>)
```

is used to read or write a data handle in a file (or to another location). This function can be used if a client is disconnected from the NES platform or to be able to share data between clients. Permission for another client to access the data could be given by setting read access permission to the file where the data handle is stored.

2.2 Data persistence

As exposed above, a programmer may explicitly determine if a data will be persistent in the platform. This is made through a flag which will set the persistence policy applied to the data. The persistence flag is stored in the data handle.

This flag is implemented as an enumerated type `grpc_persistence_mode_t` that contains the following policy values:

- VOLATILE: used when the data is not kept inside the platform after a computation (regular use of the GridRPC API),
- PERSISTENT: used when a data has to be kept on the platform. The data is not sent back to the client and potential coherency issues may arise. Moreover, the data item can be moved between servers depending on scheduling decisions.
- PERSISTENT_RETURN: used when a data is kept inside the platform but a copy is sent back to the client. This mode is useful when the client needs intermediate results. Potential coherency issues may arise. The data item can be moved between servers depending on scheduling decisions.
- STICKY: used when a data is kept inside the platform but cannot be moved between the servers. In that case the data is not given back to the client after computation. This is used if the client needs that data in the platform for a second computation on the same server.
- STICKY_RETURN: used when a data is kept inside the platform but a copy is sent back to the client. Potential coherency issues may arise. The data item stays on the server where the computation has been done.

A set function is associated to the persistence flag:

```
grpc_set_persistence_flag(grpc_data_handle_t *data_handle,
                                    FLAG)
```

3 Data management

Data Persistence is managed by the client (or by a proxy attached to it). It is responsible for declaring and deleting his data and managing its persistence in the platform.

3.1 Data declaration

To declare persistent data the client will use the `grpc_new_data` function:

```
grpc_new_data( grpc_data_handle_t* data_handle,
                    <varargs>)
```

with `data_handle` a previously created data handle. Next parameters include information on the data itself.

Some remarks concerning the use of this function:

- Data may be declared either implicitly or explicitly by the client. Implicitly, means that a call to a `grpc_call` family functions with data as parameters will return data handles for each data.

3.2 The `grpc_get_data` method

The `grpc_get_data` allows a client to gather a persistent data.

```
grpc_get_data(grpc_data_handle_t* data_handle, <varargs>);
```

The client will get the data in the buffer pointed by data.

3.3 The `grpc_free_data` method

The `grpc_free_data` allows the client to remove a data from the platform (both internal descriptions and data items).

```
grpc_free_data ( grpc_data_handle_t* data_handle );
```

4 Data Management in Diet

Let us consider the following problems: A client submit $A = B * C$ with A,B and C matrice of type double.

Note that in these examples we won't detail all the steps to build the profile. We are only interested by the use of the new data API. We can use our API in two ways : explicitly or implicitly.

- Explicit Way

```
  double *A, *B, *C=NULL;
  grpc_data_handle_t idA, idB, idC;
  grpc_function_handle_t handle;

.... // A,B initialization
  grpc_initialize();
....
  grpc_new_data_handle(&idA);
```

```
  grpc_new_data_handle(&idB);
  grpc_new_data_handle(&idC);
....
  grpc_new_data(&idA, A);
  grpc_new_data(&idB, B);
  grpc_new_data(&idC, C);
....
  grpc_function_handle_default( &handle, ‘‘MATMULT’’ );
....
  grpc_call( handle, &idA, &idB, &idC );
....
  grpc_write_data_handle( &idA,’’A matrix of double’’, fileA );
  grpc_write_data_handle( &idB,’’B matrix of double’’, fileB );
  grpc_write_data_handle( &idC,’’C matrix of double’’, fileC );
....
  grpc_finalize();
```

- Implicit Way, the id is transparently given by the system.

```
  grpc_data_handle_t idA, idB, idC;
  double *A, *B, *C=NULL;
  grpc_function_handle_t handle;

  grpc_initialize();
...
  grpc_call(handle, &idA, &idB, &idC );
...
  grpc_store_data_handle( &idA,’’A matrix of double’’, fileA );
  grpc_store_data_handle( &idB,’’B matrix of double’’, fileB );
  grpc_store_data_handle( &idC,’’C matrix of double’’, fileC );
...
  grpc_finalize();
```

A second client wants to submit a computation: $D = A + B$ with A et B stored inside the platform by the first client.

```
double *D;
grpc_data_handle_t idA, idB, idD;
grpc_function_handle_t handle;
```

```
...
  grpc_initialize();
...
  grpc_read_data_handle( &idA, fileA );
  grpc_read_data_handle( &idB, fileB );

  grpc_new_data_handle( &idD );
  grpc_new_data( &idD , D );
...
  grpc_call(handle, &idA, &idB, &idD );
...
  grpc_write_data_handle( &idD, ''D matrix of double'', fileD);
...
  grpc_finalize();
```

A client can also see the state of his data stored inside the platform or remove it

```
  double *A;
...
  grpc_get_data( &idA, A);
...
  grpc_free_data( &idA );
  grpc_free_data_handle( &idA );
...
  grpc_finalize();
```