Proposal for a Data Management API within the GridRPC

Y. Caniou, E. Caron, F. Desprez, G. Le Mahec

February 25, 2008

Data Management in the GridRPC

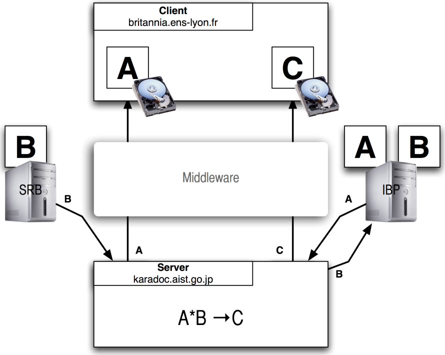
Aims of the Data Management API

- To avoid useless transfers of data
- Generic API unrelated to the data, its location, access protocol, etc.
  $\rightarrow$ Transparent access to the data from the user point of view
- Homogeneous use of different data transfer protocols
- To improve interoperability between different implementations
- The API should be compliant with SAGA API requirements

Constraints

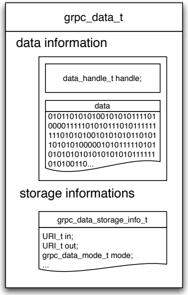
- Must be an optional improvement of GridRPC applications
- Must be in accordance with the GridRPC API
- Should be extensible to existent and future data transfer protocols

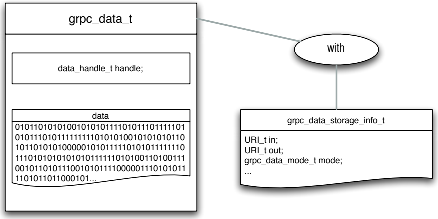
Data Management in the GridRPC: Example



GridRPC Data Types (1/2)

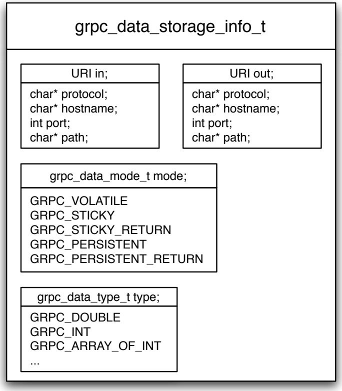
The *grpc_data_t* type contains the data or a handle on it.
The *grpc_data_storage_info_t* of a data can be in the *grpc_data_t* structure or transmitted separately by a Data Middleware.

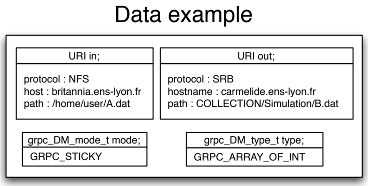




GridRPC Data Types (2/2)

The *grpc_data_storage_info_t* type contains the URI where the data can be accessed, the URI where the data will be sent after the call, the management mode and the type of the data.





Data Management Functions (1/3)

The **grpc_data_init()** Function

```
grpc_error_t grpc_data_init(grpc_data_t * data,
                            char * URI_input,
                            char * URI_output,
                            grpc_data_type_t variable_type,
                            grpc_data_mode_t storage_mode);
```

The **grpc_data_write()** Function

```
grpc_error_t grpc_data_write(grpc_data_t * data,
                             <char * server_name>);
```

The **grpc_data_read()** Function

```
grpc_error_t grpc_data_read(grpc_data_t * data);
```

Data Management Functions (2/3)

The **grpc_data_free()** Function

```
grpc_error_t grpc_data_free(grpc_data_t * data);
```

After calling *grpc_data_free()*, **data** does not reference a GridRPC data anymore. This function may be used to explicitly erase the data on a storage resource. If **data** actually contains the data, the resource is released.

Data Management Functions (3/3)

The **grpc_data_getinfo()** Function

```
grpc_error_t grpc_data_getinfo(grpc_data_t * data,
                               grpc_data_info_t info,
                               char * info);
```

The **grpc_data_load()** and **grpc_data_save()** Functions

```
grpc_error_t grpc_data_load(grpc_data_t * data,
                            char * URI_input);
grpc_error_t grpc_data_save(grpc_data_t * data,
                            char * URI_output);
```

The **grpc_data_write()** Function

Proposition for OGF'21

```
grpc_error_t grpc_data_write(grpc_data_t * data,
                             <char * server_name>);
```

Now

```
grpc_error_t grpc_data_write(grpc_data_t data, grpc_data_diffusion_t mode,
                             <char * server_name>);
```

Use of a diffusion type like

- *GRPC_UNICAST*,
- *GRPC_BROADCAST*,
- *GRPC_DEFAULT_DIFFUSION*.

The "*freeing*" Functions

Proposition for OGF'21

```
grpc_error_t grpc_data_free(grpc_data_t * data);
```

Now

```
grpc_error_t grpc_data_unbind(grpc_data_t * data);

grpc_error_t grpc_data_free(grpc_data_t * data);
```

- When the user does not need a handle anymore, but knows that the data may be used by another user for example, he can unbind the handle and the GridRPC data by calling this function without actually freeing the GridRPC data on the remote servers.
- If **data→URI_locations** is `NULL`, then the data is erased on all the locations where it is stored, else it is freed on all the location contained in the list of URI. After calling this function, **data** does not reference the data anymore.

A new data type in *grpc_data_t* and new access functions

A new label for the *grpc_data_type_t*

```
GRPC_INT, GRPC_DOUBLE, GRPC_COMPLEX,
GRPC_ARRAY_OF_INT, ...,
GRPC_MATRIX_OF_INT, ...,
GRPC_CONTAINER_OF_GRPC_DATA
```

Access Functions to Elements in a Container of *grpc_data_t*

```
grpc_error_t grpc_data_container_add( grpc_data_t * container, int rank,
                                      grpc_data_t * data );

grpc_error_t grpc_data_container_get( grpc_data_t * container, int rank,
                                      grpc_data_t * data );
```

- **container** is necessarily a grpc_data_t of type `GRPC_CONTAINER_OF_GRPC_DATA`
- **rank** is a given integer which acts as a key index
- **data** is the data that the user wants to add in or get from the container

⟶ Getting the data does not remove the data from **container**

⟶ Container management is free of implementation

Usage of Containers

Whatever the management of the container by the GridRPC Data Management Middleware (list, array, etc.)

Correct and Portable Client Code

```
grpc_data_t pool, d1, d2;

/* Data initializations */
grpc_data_init(&d1,
               "IBP://kaamelott.cs.utk.edu/1212#A.dat/ReadKey/READ",
               "NFS://britannia.ens-lyon.fr/home/user/A.dat",
               GRPC_DOUBLE, NULL);
grpc_data_init(&d2,
               "LOCAL_MEMORY://britannia.ens-lyon.fr/&A",
               "LOCAL_MEMORY://karadoc.aist.go.jp",
               GRPC_DOUBLE, GRPC_STICKY);
grpc_data_init(&pool,NULL,NULL, GRPC_CONTAINER_OF_GRPC_DATA, NULL);
/* ... */
grpc_data_container_add(pool, 1, d1);
grpc_data_container_add(pool, 2, d2);
```

New Functions?

Goals

- To manage asynchronous transfers
- Multiple servers parallel writings

Example of Needed Functions

```
grpc_error_t grpc_data_transfer_wait(grpc_data_t * data, ???);
```

Because some transfers can be synchronous or asynchronous?
$\longrightarrow$ The user can know for sure that the data is in place.

Conclusion & Future Works

In Brief

- Simple API for data management with only 10 functions
- Allowing a simple and powerful data management from the API
- Taking into account many use cases (all?)

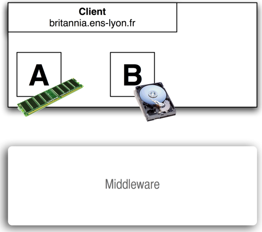
Roadmap

- How to manage multiple data repositories?
- Implementation
- GridRPC data management interoperability
  - Work on synchronous/asynchronous transfer
  - New document
  - Interoperability testing for the GridRPC data API specification
  - Error codes to be defined

5 GridRPC Data Management: Using the API
- Example 1
- Example 2

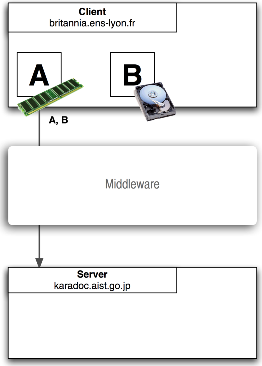
Simple RPC Call With Input and Output Data

```
grpc_data_init(&dhA, "LOCAL_MEMORY://britannia.ens-lyon.fr/&A", NULL, GRPC_DOUBLE,
                    GRPC_VOLATILE);
grpc_data_init(&dhB, "NFS://britannia.ens-lyon.fr/home/user/B.dat", NULL, GRPC_DOUBLE,
                     GRPC_VOLATILE);
grpc_data_init(&dhC, NULL, "NFS://britannia.ens-lyon.fr/home/user/C.out", GRPC_DOUBLE,
                    GRPC_VOLATILE);
```



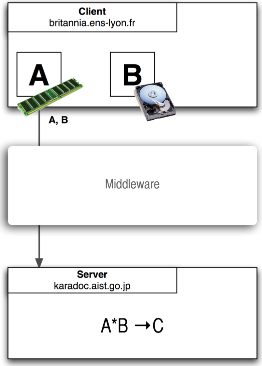
Simple RPC Call With Input and Output Data

grpc_function_handle_init(handle1, "karadoc.aist.go.jp", "*");



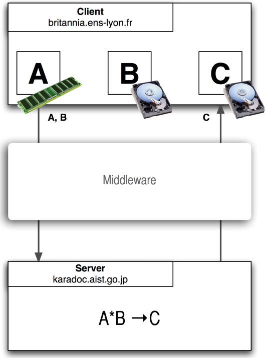
Simple RPC Call With Input and Output Data

```
grpc_call(handle1, dhA, dhB, &dhC);
```



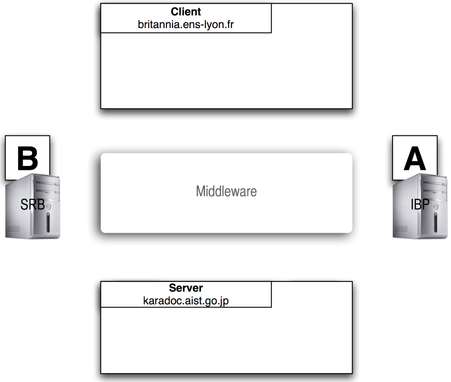
Simple RPC Call With Input and Output Data

Output data **C** is sent back to the client.



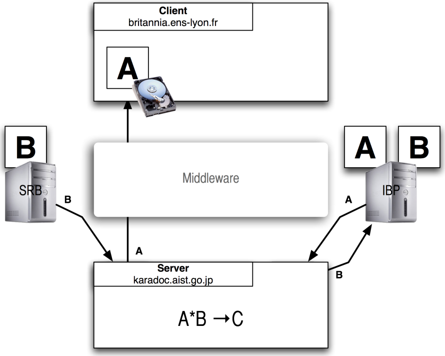
Storage with External Storage Resources

```
grpc_data_init(&dhA, "IBP://kaamelott.cs.utk.edu/1212#A.dat/ReadKey/READ",
               "NFS://britannia.ens-lyon.fr/home/user/A.dat", GRPC_DOUBLE,
                GRPC_VOLATILE);
grpc_data_init(&dhB, "SRB://carmelide.ens-lyon.fr/COLLECTION/Simulations/B.dat",
               "IBP://kaamelott.cs.utk.edu/1213#B.dat/WriteKey/WRITE", GRPC_DOUBLE,
               GRPC_VOLATILE);
grpc_data_init(&dhC, NULL, "NFS://britannia.ens-lyon.fr/home/user/C.out", GRPC_DOUBLE,
               GRPC_VOLATILE);
```



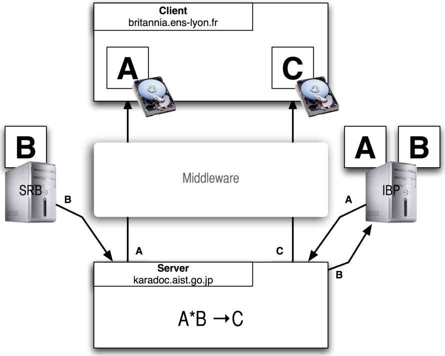
Storage with External Storage Resources

grpc_call(handle1, dhA, dhB, &dhC);



Storage with External Storage Resources

Output data **C** is sent back to the client.


Client
britannia.ens-lyon.fr
A
C
B
SRB
B
Middleware
A
B
IBP
A
A
C
B
Server
karadoc.aist.go.jp
A*B →C