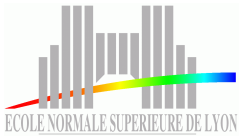
OGF standardized GridRPC Data Management API: description, performances and use case

Y. Caniou, G. Le Mahec, E. Caron, H. Nakada



Oct. 10 2012

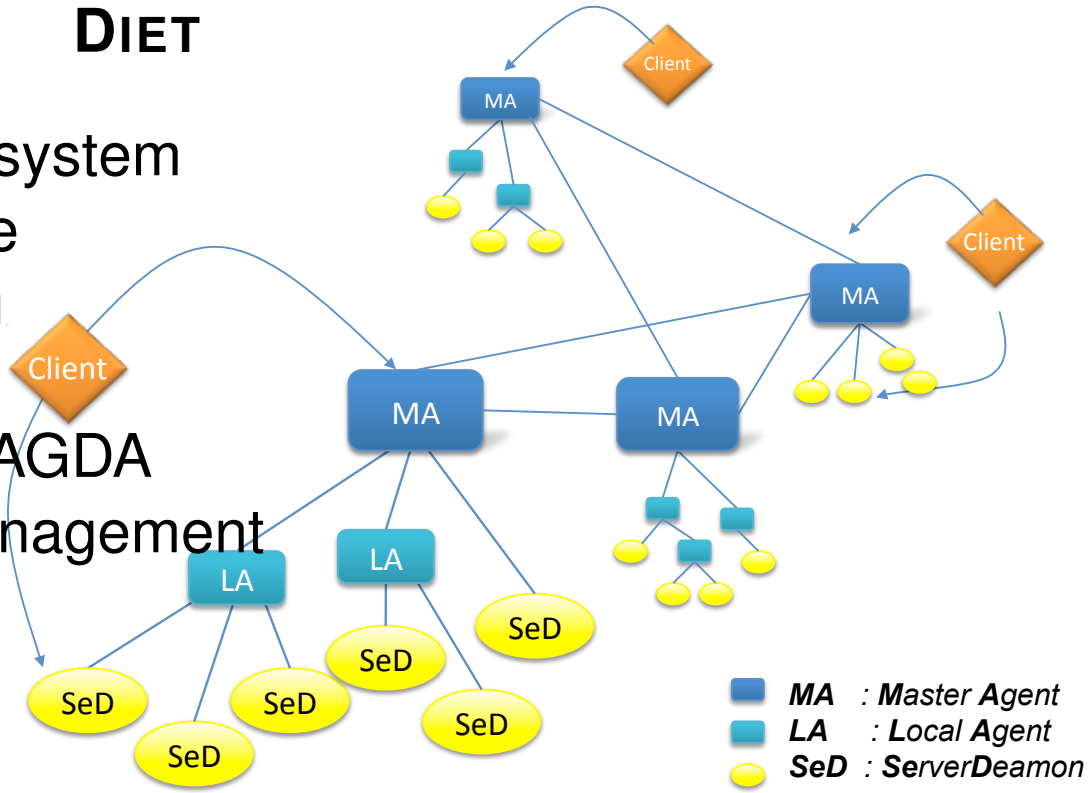
GridRPC middleware – 1/3

- DIET developped at Lyon, France: Univ. de Lyon and SysFera
- NetSolve/GridSolve developped at Univ. of Tennessee, USA
- Ninf developped at AIST, Japan
- OmniRPC developed at Univ. of Tsukuba, Japan
- XtremWeb by INRIA, France

GridRPC middleware – 2/3

DIET

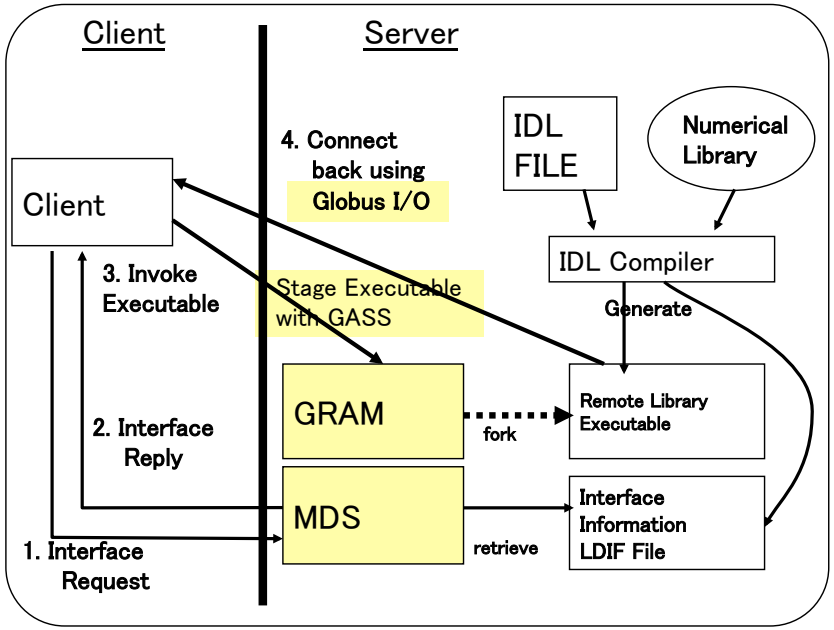
- Lightweight GridRPC system
- high perf. and scalable
- Distributed scheduling
- .. application specific!
- Data management: DAGDA
- Dynamic workflow management
- Firewall forwarders
- LRMS management
- DIET Cloud
- and more...



v2.8.1 in 2012, July @ `http://graal.ens-lyon.fr/diet`

GridRPC middleware – 3/3

Ninf



- Simple GridRPC system
  → no scheduling
- Relies on Globus
- and standards: GridFTP, etc.
- Inkoke
  - Naregi
  - Unicore
  - ssh servers

v5.1.0 @ `http://ninf.apgrid.org/`

What is GridRPC? 1/2

GridRPC

- Call procedures / functions which reside on remote sites
- Easy to use just call procedures / functions
- No parameter marshaling are required

Advantage

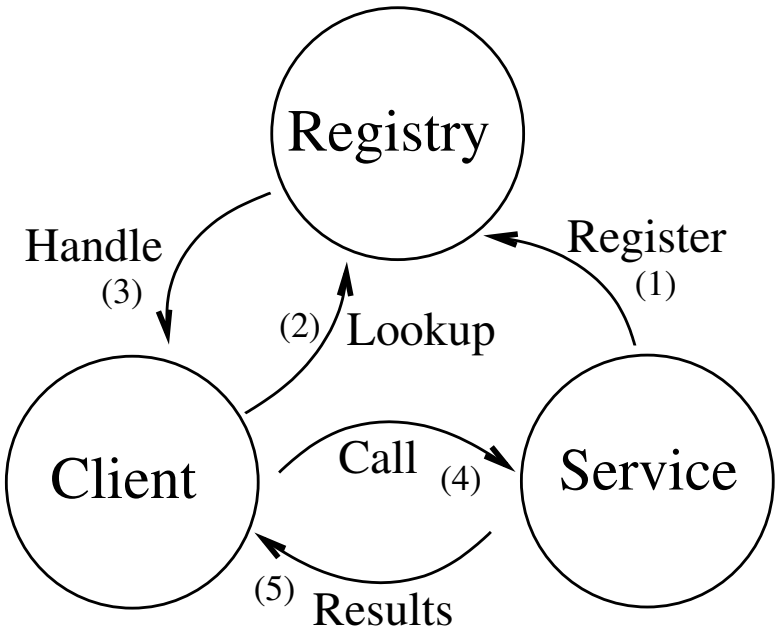
- Simple, easy to use
- Parallel processing by simultaneous invocations
- Optional scheduling to optimize one or numerous criteria

Example Program Segment

```
int n=5, incx=1, incy=1, status;
double ns_result = 0.0;
double dx[] = {10.0, 20.0, 30.0, 40.0, 50.0};
double dy[] = {60.0, 70.0, 80.0, 90.0, 100.0};
grpc_function_handle_t handle;

/* Initialize GridRPC module */
grpc_initialize(NULL);

/* Create GridRPC function handle */
grpc_function_handle_default(&handle, ``ddot'');

/* Make the call */
status = grpc_call(&handle, &n, dx, &incx, dy, &incy,
                   &ns_result);
```

What is GridRPC? 2/2

API defined by the OGF GFD-R.052 standard.



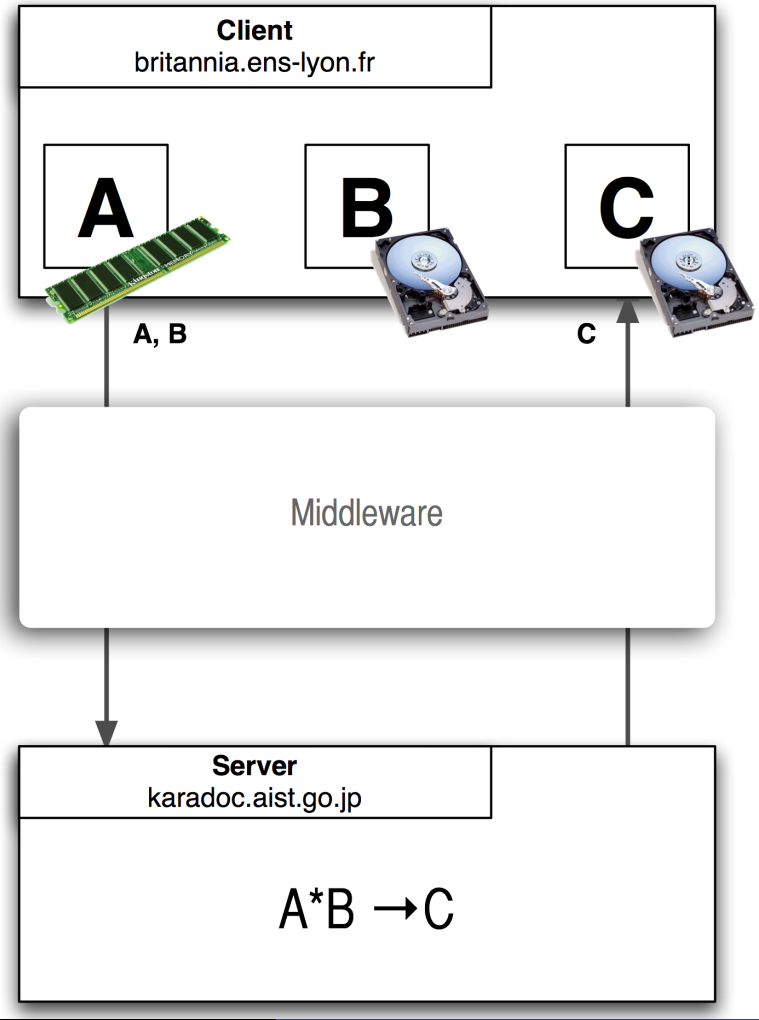
Making GRPC calls

- `grpc_call(srv, params)`
- `grpc_call_async(srv, params)`

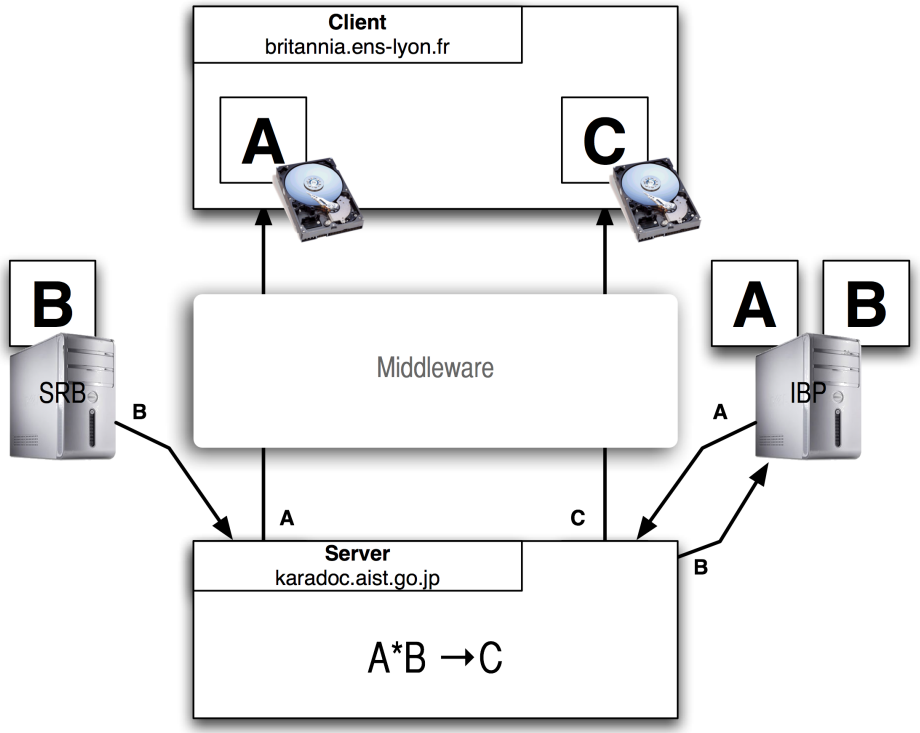
Limitations

- Code portability
- Feasibility
- Transfer management
- ... # of params

On an example of what is now



On an example of what we want



- Code portability! Computation feasibility!
- Performance with migration, stickiness, persistence.
- Additionnal feature: number of parameters, extensibility!

Data Management in the GridRPC

Aims of the Data Management API

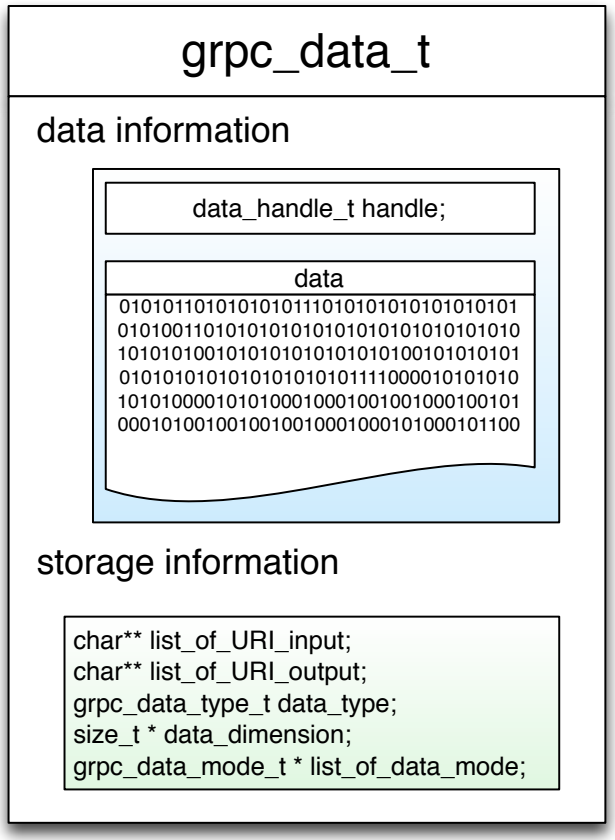
- Avoid useless transfers of data
- Generic API unrelated to the data, its location, access protocol...
  $\longrightarrow$ Transparent access to the data from the user point of view
- Homogeneous use of different data transfer protocols
- Improve interoperability between different implementations

Constraints

- Must be an optional improvement of GridRPC applications
- Must be in accordance with the GridRPC API

GridRPC Data Type

The `grpc_data_t` type contains the data or a handle on it.



protocol://[user:password@]**hostname**[:port][/data_path]**/data**

grpc_data_type_t

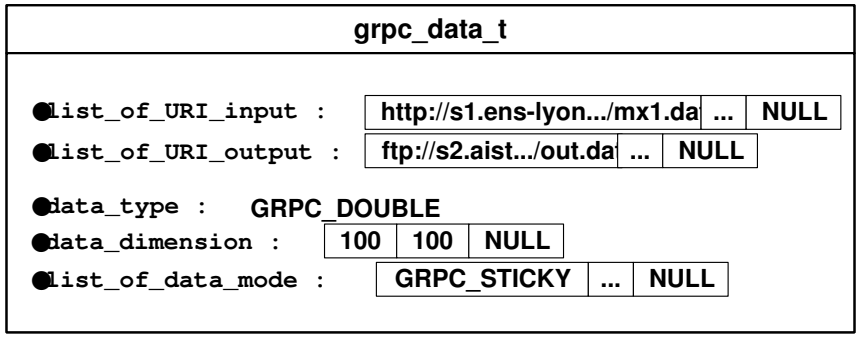
- GRPC_DOUBLE
- GRPC_INT
- CONTAINER_OF_GRPC_DATA
- GRPC_STRING
- ...

grpc_data_mode_t

- GRPC_VOLATILE
- GRPC_STRICTLY_VOLATILE
- GRPC_STICKY
- GRPC_UNIQUE_STICKY
- GRPC_PERSISTENT

GridRPC Data example

In this example, the grpc_data_t was initialized to use a matrix 100×100 of doubles located on an http server. The matrix is stored on an external ftp server with the STICKY persistence.



Data Management functions – 1/7

The **grpc_data_init()** function

```
grpc_error_t
grpc_data_init(grpc_data_t * data,
               const char ** list_of_URI_input,
               const char ** list_of_URI_output,
               const grpc_data_type_t data_type,
               const size_t * data_dimension,
               const grpc_data_mode_t * list_of_storage_mode);
```

This function initializes the GridRPC data with a specific data.
This data may be available locally or on a remote storage server. Both identifications can be used.
GridRPC data referencing input parameters must be initialized with identified data before being used in a `grpc_call()`.
GridRPC data referencing output parameters can be initialized to NULL for an empty list.

Data Management functions – 2/7

The **grpc_data_transfer()** function

```
grpc_error_t
grpc_data_transfer(grpc_data_t * data,
                   const char ** list_of_URI_input,
                   const char ** list_of_URI_output,
                   const grpc_data_mode_t * list_of_output_mode);
```

A user may want to be able to transfer data while computations are done. For example, if a computation can begin as soon as some data are downloaded but needs all of them to finish, the management of data must use **asynchronous mechanisms** as default behavior. Then, this function initiates the call for the transfers and returns immediately after.

Data Management functions – 3/7

The **grpc_data_wait()** function

```
grpc_error_t
grpc_data_wait(const grpc_data_t ** list_of_data,
               grpc_completion_mode_t  mode,
               grpc_data_t ** returned_data);
```

Depending on the value of **mode** (`GRPC_WAIT_ALL` or `GRPC_WAIT_ANY`), the call returns when all or one of the data listed in **list_of_data** is transfered, which means that for a given data, all transfers involved for the input *or* output part are finished.

Data Management functions – 4/7

The **grpc_data_unbind()** function

```
grpc_error_t
grpc_data_unbind(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. After calling this function, `data` does not reference the data anymore.

Data Management functions – 5/7

The **grpc_data_free()** function

```
grpc_error_t
grpc_data_free(grpc_data_t * data, const char ** URI_locations);
```

If **URI_locations** is `NULL`, then the data is erased on all the locations where it is stored, else it is freed on all the locations contained in the list of URI.
After calling this function, **data** does not reference the data anymore.

Data Management functions – 6/7

The **grpc_data_getinfo()** function

```
grpc_error_t
grpc_data_getinfo(const grpc_data_t * data,
                  grpc_data_info_type_t info_tag,
                  const char * URI,
                  char ** info);
```

The kind of information that the function gets is defined by the **info_tag** parameter. A server name can be given to get some data information dependent on the location of where is the data (like `GRPC_STICKY`). **info** is a `NULL`-terminated list containing the different available information corresponding to the request.

Data Management functions – 7/7

The **grpc_data_load()** and **grpc_data_save()** functions

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

These functions are used to load/save the data descriptions. Even if the GridRPC `data` contains the data in addition to metadata management information (data handle, size, type, etc.), only data information have to be saved in the location. The format used by these functions is let to the developer's choice. The way the information are shared by different middleware is out of scope of this document and should be discussed in an interoperability recommendation document.

Mappings of memory locations to keywords

Mapping functions

```
grpc_error_t
grpc_data_memory_mapping_set(const char * key, void * data);

grpc_error_t
grpc_data_memory_mapping_get(const char * key, void ** data);
```

If he wants to use a data which is in memory, the user must provide some name in the URIs in the input or output fields which has to be understood by the GridRPC Data Management layer in the GridRPC system, in addition of the use of the *memory* protocol.

For example, `grpc_data_memory_mapping_set()` is used to make the relation between a data stored in memory and a `grpc_data_t` data when the *memory* protocol is used: it records the keyword that will be used in URIs, for example during the initialization of the data.

New data type in `grpc_data_t`, and access functions

A new label for the `grpc_data_type_t`

`GRPC_BOOL`, `GRPC_INT`, `GRPC_DOUBLE`, `GRPC_COMPLEX`, `GRPC_STRING`, `GRPC_FILE` `GRPC_CONTAINER_OF_GRPC_DATA`

Access functions to elements in a container of `grpc_data_t`

```
grpc_error_t
grpc_data_container_set(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

Aims

Proof of concept of

- Implementation of a GridRPC DM library
- ... showing that DM API extension is key solution
- Performances!
- Cooperation to a unique resolution
  - across different domains
  - using 2 different middleware frameworks

Grid testbed

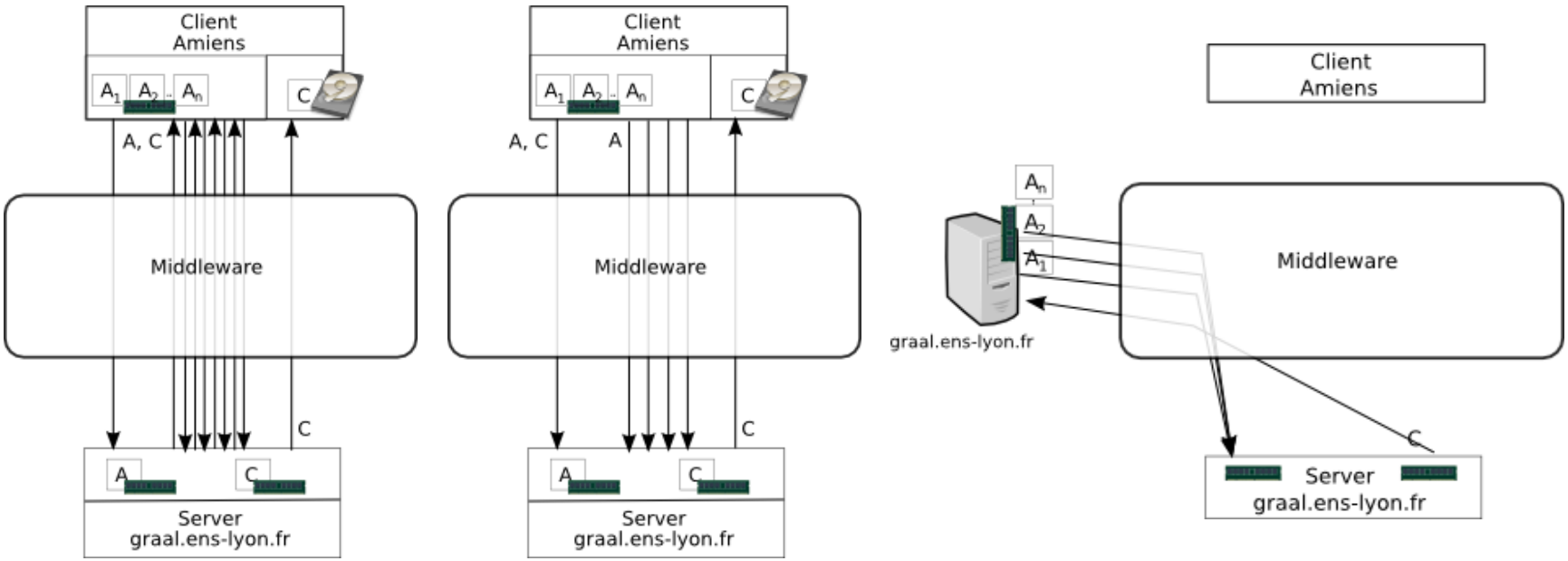
- `aist`: VM with 8 cpus 2.4 GHz, around 22GB RAM and running the linux kernel 3.0.0-15 32 bits, located in Tsukuba (Japan) and accessible on the SINET network. It is used as a **GridRPC server**.
- `graal.ens-lyon.fr`: 16 4core cpu 2.93 GHz Intel Xeon X5570, with 32GB RAM and running the linux kernel 2.6.18-27 64 bits, located in Lyon (France) and accessible on the RENATER network. It is used as a **GridRPC server** and as a **HTTP server**,
- `client`: MacBook Pro core 2 duo 2.4 GHz with 4GB RAM, running the Mac OS 10.6.8 64 bits, located in Amiens (France) and accessible through a ADSL 2+ connection. It is the **GridRPC client**.

• Bandwidth: 24 trsansfers of
- 2MB matrices if `client`
- 32MB matrices otherwise
• Latency 300msec
• RENATER-GEANT2-SINET

↱	aist	graal	client
aist	-	8271.20	1013.73
graal	11260.01	-	2454.07
client	569.52	739.23	-

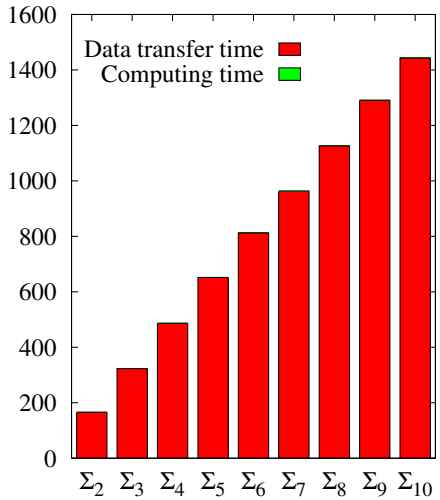
Scenario 1, 3 experiments

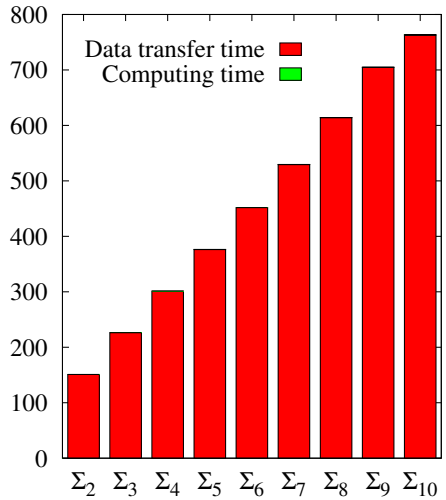
Compute $\sum Ai$, 1000×1000 matrices

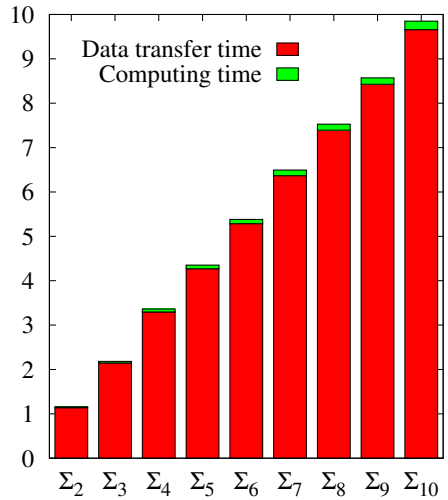


Scenario 1, results

Compute $\sum Ai$, 1000×1000 matrices

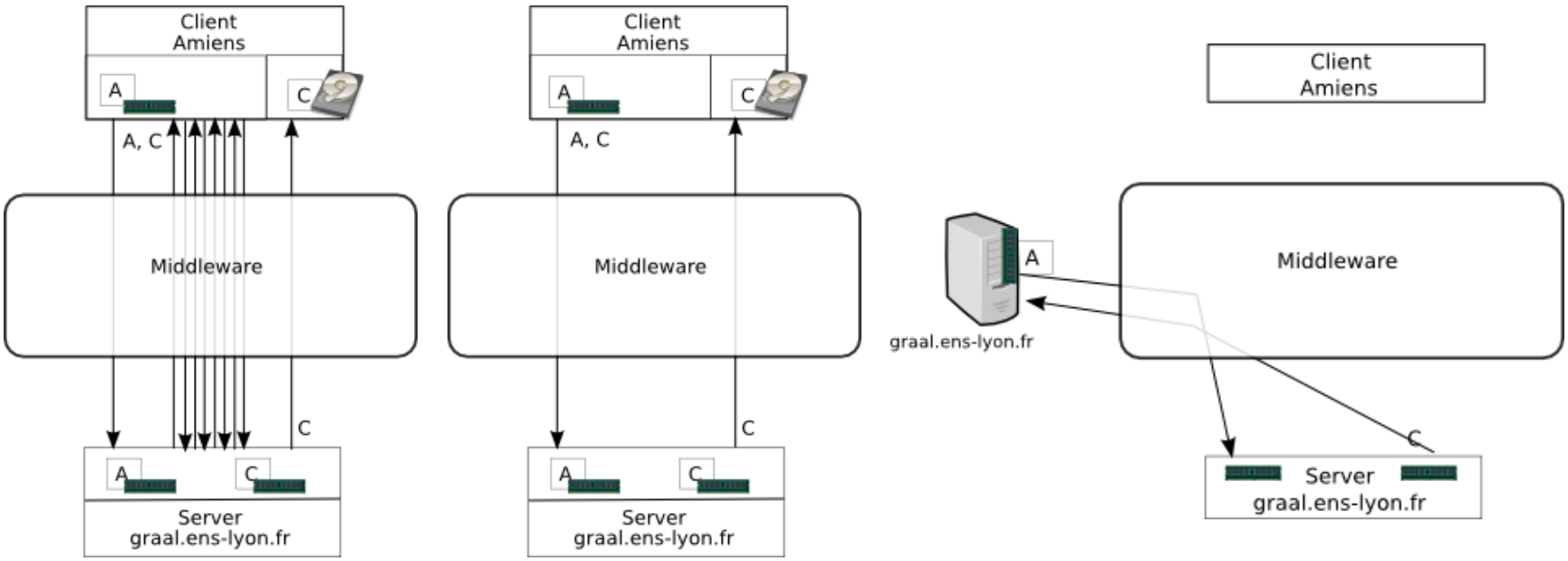






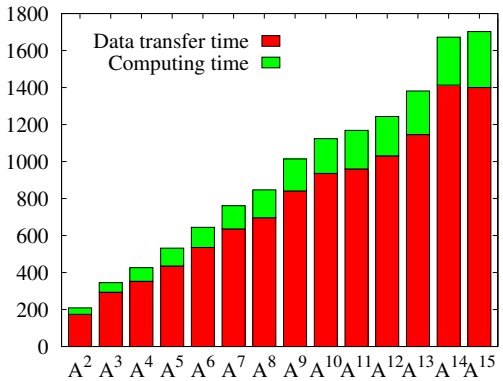
Scenario 2, 3 experiments

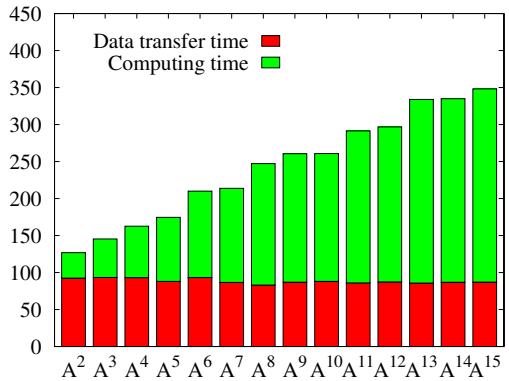
Compute A^n, 1000×1000 matrix

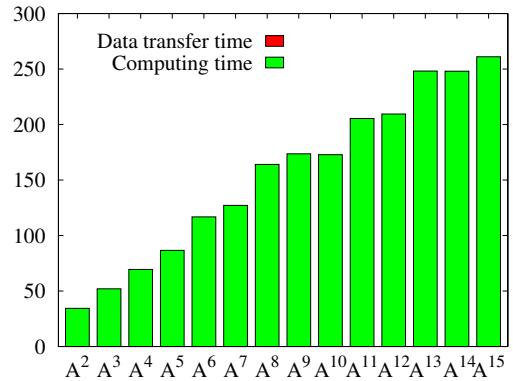


Scenario 2, results

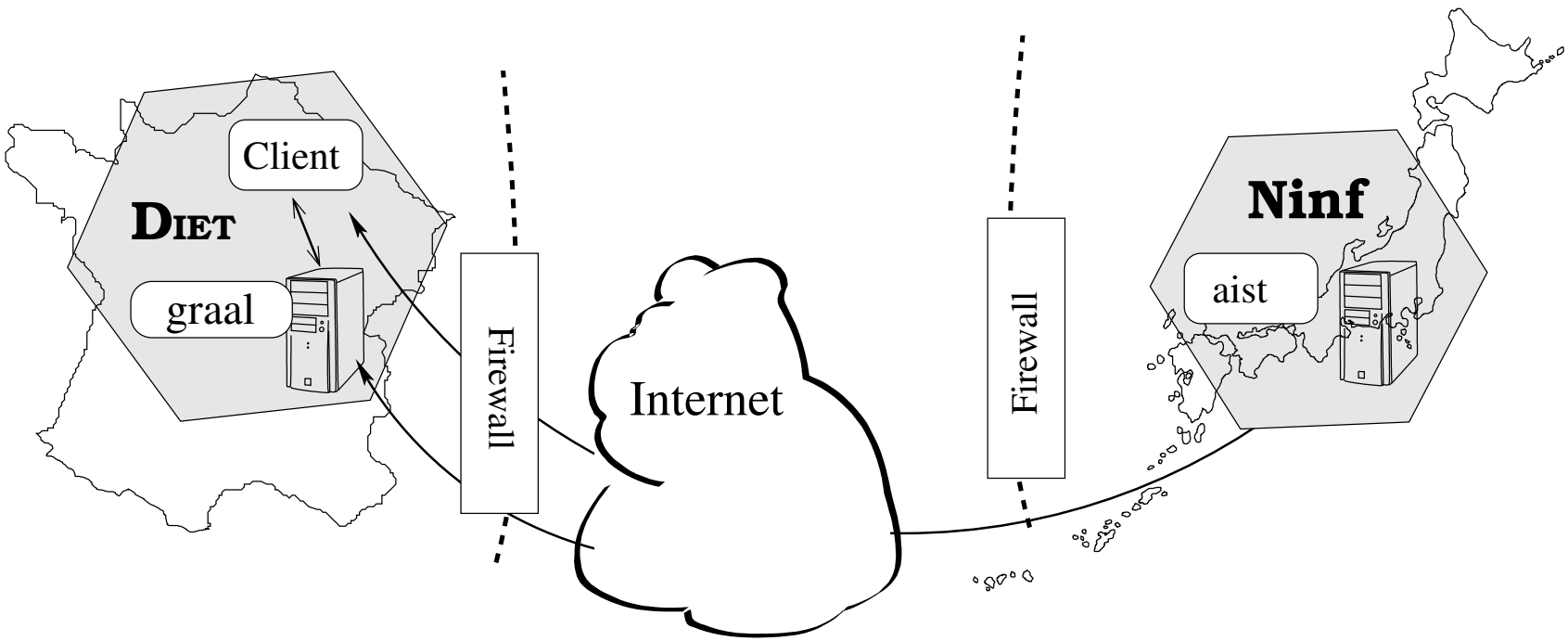
Compute A^n, 1000×1000 matrix







Grid testbed



DIET- Ninf collaboration! (^:^)v

Grid user service

Compute $E = (A * B)^3$, 1000×1000 matrices

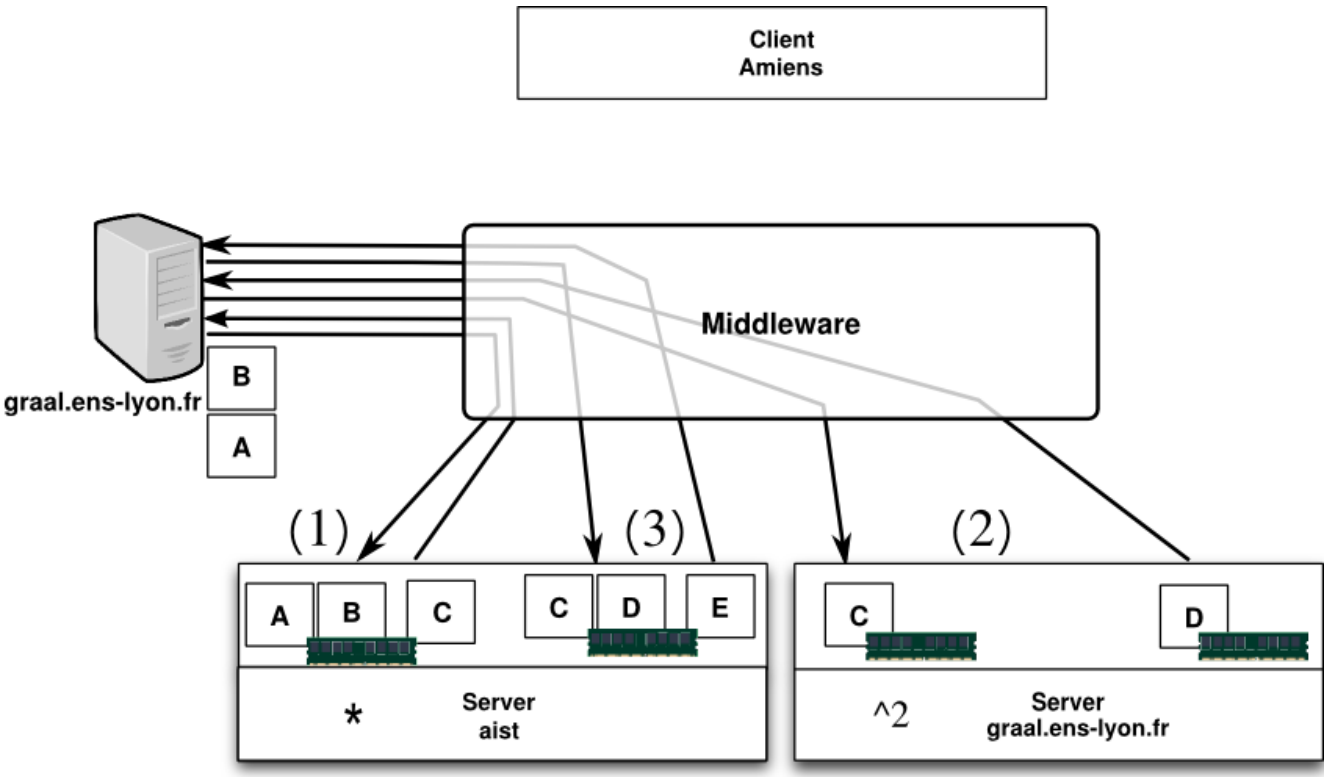
with

- Service "$*$" available on Ninf
- Service "^{2}" available on DIET

How to process?

- Using Ninf alone: $((((A * B) * A) * B) * A) * B$
- Using DIET and Ninf
  1. $C = A * B$ on Ninf
  2. $D = C^2$ on DIET
  3. $E = C * D$ on Ninf

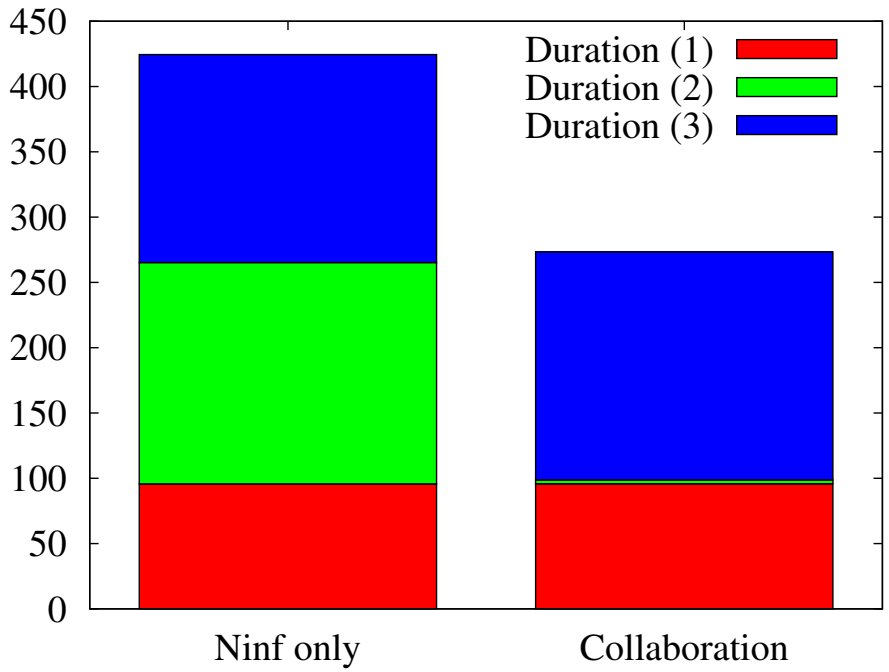
Workflow on DIET- Ninf platform



DIET- Ninf collaboration! (^ ; ^) v

Collaboration to a unique workflow, results!

Experiment	Duration (1)	Duration (2)	Duration (3)
Ninf only	95724	169513	159266
Collaboration	95780	2966	174570



Implementation details

- One GridRPC glue, one client!
  - Service handle: `DIET:MatMult` or `NINF:MatMult`
  - Convert args to corresponding middleware API
  - at the moment, ssh call for Ninf
- Transfer protocols
  - Use HTTP POST, using open source libCURL
  - 10line PHP script on HTTP server
  - HTTP **and** `memory` protocols

Objectives

https://forge.mis.u-picardie.fr/projects/gridrpcdm/

The objective of this library is to provide a complete implementation of the API:

- Easily extensible to all GridRPC middleware
- Providing interoperability between the middleware implementations without modifying them
- Easily extensible to all data transfer protocols
- Providing data transfer protocols to middleware without any modification

Possible Middleware

To be partially compatible with the library, a middleware must:

- Manage characters string as input parameters of a service
- Be usable from C/C++ code

To be fully compatible with the library, a middleware must:

- Be able to call C/C++ code on the server part

Partially compatible: the library manages all the transfers from the client.
Fully compatible: the library can manage transfers remotely using specific predefined services.

Possible Transfer Protocols

A data transfer protocol can be used by the library if:

- It can get data from a *source* server using C/C++ call
- It can put data to a *destination* server using C/C++ call

Library Extensibility and Portability

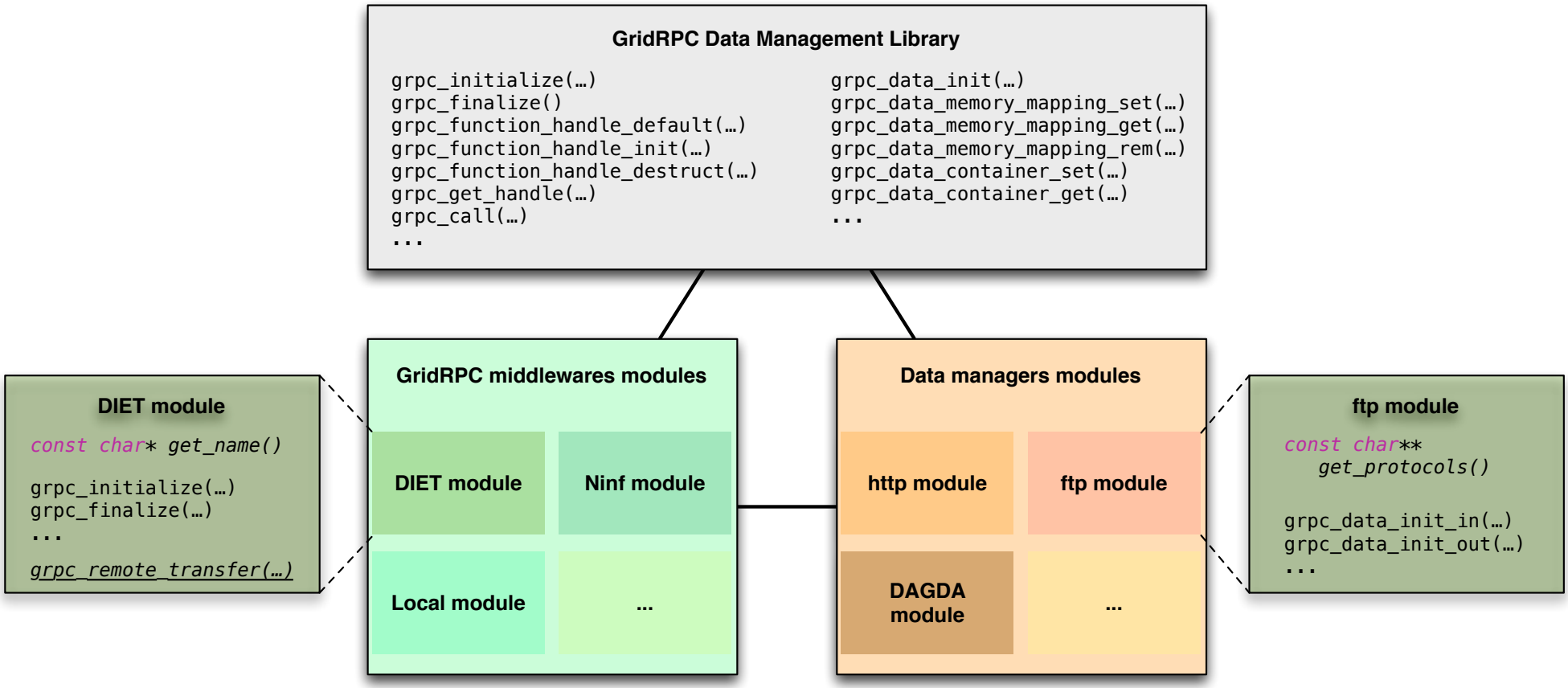
Extensibility

The library is designed to be easily extensible to different middleware and different protocols by providing a loadable *module* system.

Portability

The library uses standard C/C++ routines and the Boost library. Only the loadable module support is using the non-portable POSIX *dl* library.

Library General Architecture



Modules Implementation

The library was designed to ease the implementation of modules for different middleware and data transfer protocols. It manages the middleware and protocols interactions, requiring only some simple functions developments.

The library provides as much as possible default implementations and utility functions to help the modules developers.

Modules Implementation

grpc_initialize(…)

```
...
for (m = middlewares.begin(); m!=middlewares.end();
     ++m) {
  m->initialize();
  ...
}
...
```

grpc_data_init(…)

```
...
dm_module = dm_modules.find(protocol(uri_in));
dm_module->data_init_in(…);
...
```

grpc_finalize()

```
...
for (m = middlewares.begin(); m!=middlewares.end();
     ++m) {
  m->finalize();
  ...
}
...
```

grpc_data_get(…)

```
...
dm_module = dm_modules.find(protocol(uri));
dm_module->grpc_data_get(…);
...
```

...

grpc_default_remote_transfer(…)

```
grpc_data_init(source, …);
grpc_data_init(dest, …);
…
grpc_function_handle_init(hdl, "remote-TAG");
grpc_call(hdl, source, dest, …);
```

GridTLSE 1/2

http://gritlse.org

Aims

An expert site for linear algebra aims to provide tools & software for sparse matrices: provides user assistance to evaluate and choose the best solver for given problems, helps to set the appropriate values of the input parameters that control the efficiency of the selected solver (Sparse Direct Solvers currently available are MA48, MA49, MUMPS, SuperLU, UMFPack).

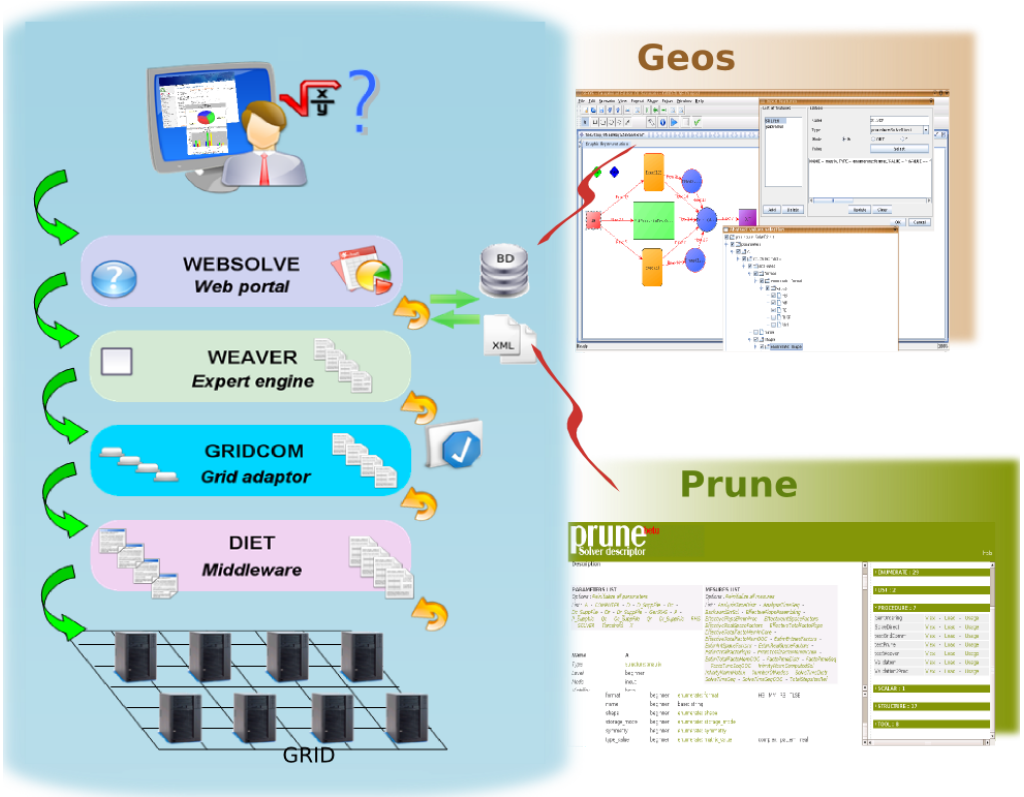
Functionalities

- Consult database of collect. of sparse matrices & download matrix files
- Use the site as a platform for cooperative work
- Upload matrices into the site
- Quickly evaluate sparse direct solvers & obtain statistics on solving sparse linear systems $Ax = b$.

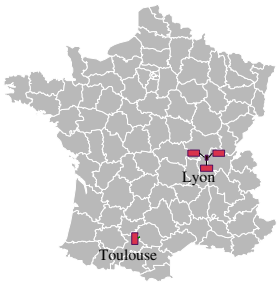
GridTLSE 2/2

GridTLSE relies on

- Websolve
  - Geos
  - Prune
- Weaver
- GridCom
- → DIET



Testbed



composed of the machine running the GridTLSE web site (and its matrices collection) in Toulouse, and a cluster in Lyon, with the MA and the client launched on the frontal, and two SEDs running on two of its computing resources.

Expertises, Matrices

Each expertise uses one of the 5 matrices. For each expertise, we consider 3 test cases, each containing two calls to a solver requesting the use of the same matrix.

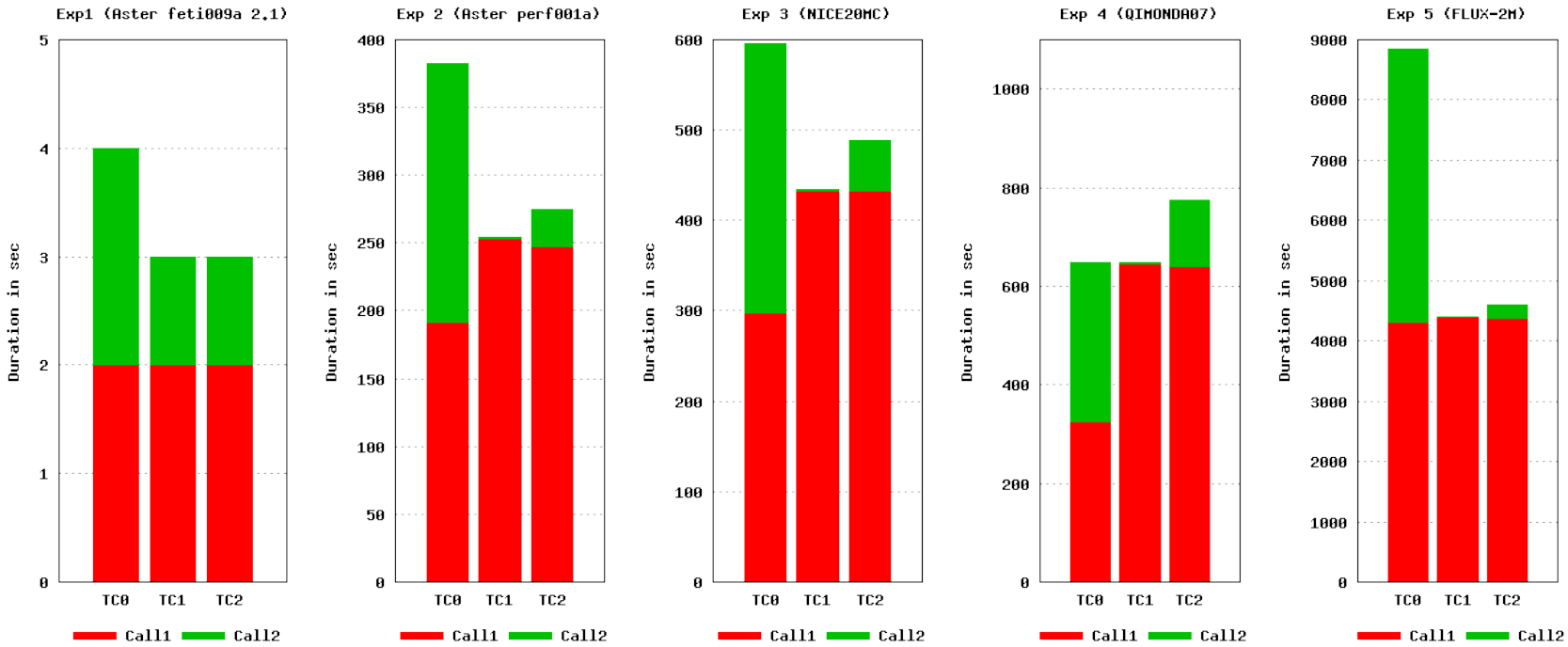
Matrix	raw	gzip	lzma
*Aster_feti*009*a*_2.1			
Size	1 563	353	186
Time	-	0	0
*Aster_perf*001*a*			
Size	534 224	131 813	56 748
Time	-	13	32
*NICE*20*MC*			
Size	895 087	205 756	70 807
Time	-	20	34
*QIMONDA*07			
Size	2 195 699	214 549	97 850
Time	-	34	69
FLUX − 2*M*			
Size	16 007 994	3 000 860	1 849 934
Time	-	325	1068

Transfers Description

Test case 0: Do not use DAGDA. Transfers are explicitly made: when a service is executed on a SED, it downloads the gziped matrix from the GridTLSE website. Hence, two transfers occur and $D_0 = (T_{Mc}/B_1 + d) * 2$

Test case 1: Uses DAGDA. The client downloads the gziped matrix from the GridTLSE site, uncompresses and registers it into DAGDA by sending the uncompressed matrix to the MA. Then the SED downloads the matrix from the MA. When the second call is performed on the same SED, there is still a copy of the matrix on the SED, so the data is immediately available and $D_1 = (T_{Mc}/B_1 + d) + T_M/B_2$

Test case 2: Uses DAGDA. The expertise is the same than for Test cases 0 and 1, but the second execution is conducted on a different computing resource, thus an additional transfer from the MA (but more generally DAGDA is able to choose the best location from where to download data depending on internal statistics and monitoring). Hence we have $D_2 = (T_{Mc}/B_1 + d) + 2 * T_M/B_2$


Exp1 (Aster feti009a 2.1)
Exp 2 (Aster perf001a)
Exp 3 (NICE20MC)
Exp 4 (QIMONDA07)
Exp 5 (FLUX-2M)
Duration in sec
TC0
TC1
TC2
Call1
Call2

Conclusion & Future Works

In Brief

- Simple API for data management with only 12 functions
- Allowing a simple and powerful data management from the API
- Taking into account many use cases (all?)
  → send us your case!

Performances

- Portability, feasibility
- Heterogeneous architectures
- Grid middleware collaboration
  → across different administrative domains
  → **transparently!**
- Answers requirements, and more!
  . . . and only the beginning

Conclusion & Future Works

Roadmap

- More protocols, more tests, ...
- GridRPC data management interoperability
  - Interoperability testing for the GridRPC data API specification
    $\rightarrow$ New document for OGF
  - Handle data format (OGF DFDL language?)
- Complementary topics
  - meta-scheduling
  - security

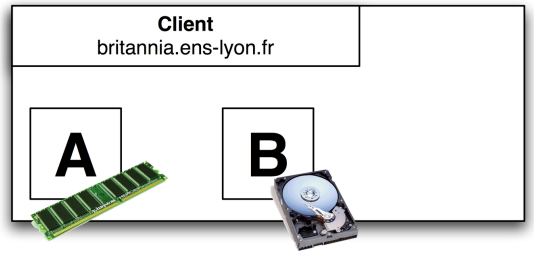
We are eager to get different use-cases and users feedbacks...

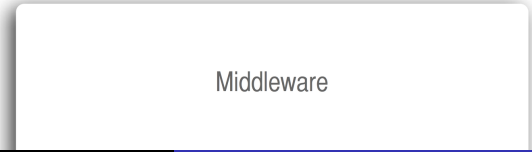
7 GridRPC Data Management: using the API

- A simple example

Simple RPC call with input and output data

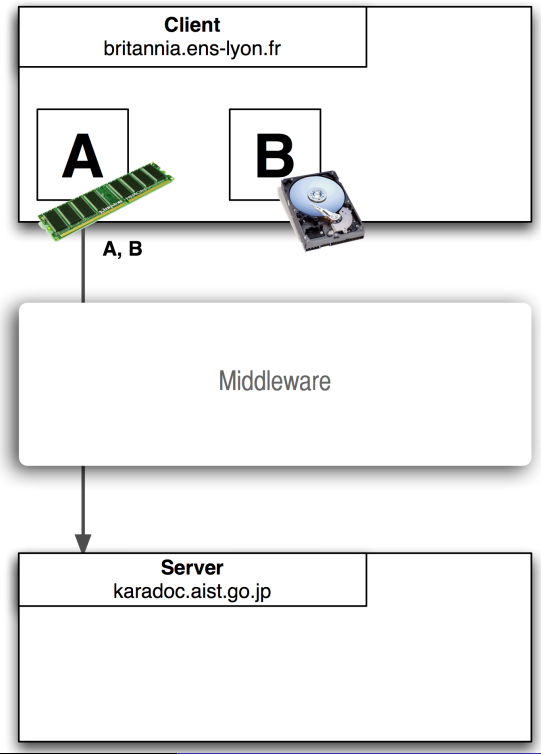
```
grpc_data_init(&dhA,
                (const char * []){"memory://britannia.ens-lyon.fr/&A", NULL},
                NULL,
                GRPC_DOUBLE, (size_t []){3, 3, 0},
                (grpc_data_mode_t []){GRPC_VOLATILE, GRPC_END_LIST});
grpc_data_init(&dhB,
                (const char * []){"nfs://britannia.ens-lyon.fr/home/user/B.dat", NULL},
                NULL,
                GRPC_DOUBLE, (size_t []){3, 3, 0},
                (grpc_data_mode_t []){GRPC_VOLATILE, GRPC_END_LIST});
grpc_data_init(&dhC,
                NULL,
                (const char * []){"nfs://britannia.ens-lyon.fr/home/user/C.out", NULL},
                GRPC_DOUBLE, (size_t []){3, 3, 0},
                (grpc_data_mode_t []){GRPC_VOLATILE, GRPC_END_LIST});
```





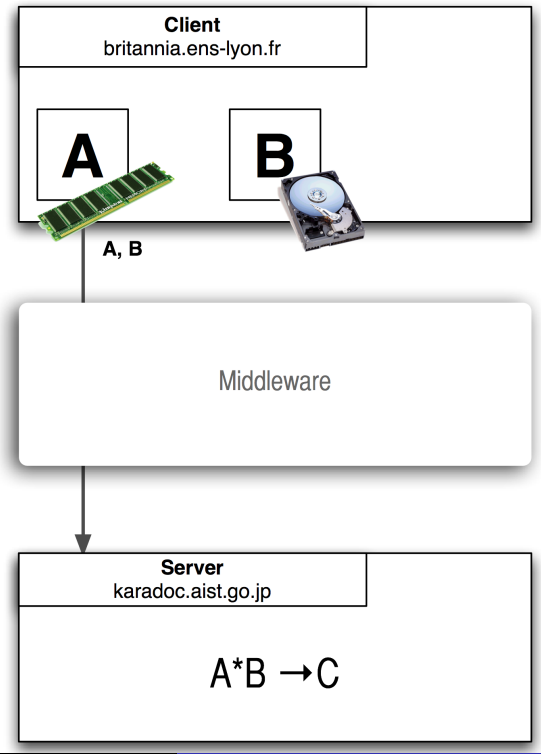
Simple RPC call with input and output data

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.