

## Preliminary Study of A Task Farming API over The GridRPC Framework

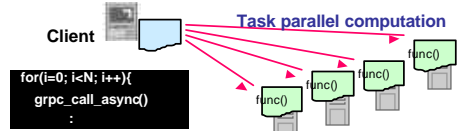
**Yusuke Tanimura, Hidemoto Nakada  
Yoshio Tanaka, and Satoshi Sekiguchi**

**National Institute of Advanced Industrial  
Science and Technology (AIST)**



## GridRPC

- **RPC mechanism for the grid computing**
- **A programming model for the grid applications**
  - ▶ The API is being standardized in the working group of GGF.
    - ⊗ **An end-user API** is defined and available on two systems.
    - ⊗ **A middleware API** is being discussed.
  - ▶ Reference implementation: **Ninf-G** and **GridSolve**
  - ▶ Task parallel programming with well-known RPC semantics
  - ▶ Easy to treat a server-side fault because of 1-N model



## GridRPC programming

### ■ A typical GridRPC program (with the end-user API)

```

grpc_init()                ▶ Library initialization
grpc_function_handle_init(handle, host, func) ▶ Create a handle
:
for(i=0; i<N; i++)        _____(Loop)
  grpc_call_async(handle, A, B, C) ▶ Invoke asynchronous RPC
:
grpc_wait_all()           ▶ Wait all RPCs are completed
:
grpc_function_handle_destruct() ▶ Destruct each handle
grpc_finalize()           ▶ Library finalization
  
```

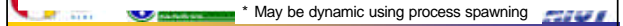
### ■ Save time to develop applications using GridRPC API

- ▶ Machine heterogeneity is wrapped by the library.
- ▶ Data communication is wrapped by the library.
- ▶ A client program written in the standard API is portable.



## GridRPC v.s. MPI

	GridRPC	MPI
parallelism	task parallel	data parallel
model	client/server	SPMD
API	GridRPC API	MPI
co-allocation	dispensable	indispensable
fault tolerance	good	poor (fatal)
private IP nodes	available	unavailable
resources	can be dynamic	static*
others	easy to gridify existing apps.	well known seamlessly move to Grid



## Case studies until today

### ■ Tests with real science

- ▶ **Scalability**: Multi-sites simulation using 500 CPUs in SC'03
- ▶ **Long-time execution**: Routine-based experiment on the Asia Pacific Grid testbed for 3 months in 2004
  - ⊗ The client could continue to run for a week.
- ▶ **Scalability + Long-time execution**
  - ⊗ Simulation with 1800 CPUs for 10 hours just before SC'04
  - ⊗ Simulation with 768 CPUs for 4.7 days just before SC'05

### ■ Lessons learned

- ▶ Needed to implement error handling, enabling heartbeat, background recovery, and remote re-initialization
- ▶ More sophisticated API could be provided.
  - ⊗ Automation in task assignment and fault recovery
  - ⊗ Higher-level APIs rather than the end-user API of the GridRPC



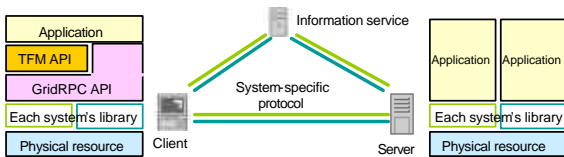
## Our purpose

- **Design and implement a high-level API (TFM API) library to develop production-quality applications**
- **Give feedback to standardizing process**
  - ▶ Show essential functionality for implementing a high-level API library
- **Focus on Task Farming (TFM)**
  - ▶ Execute a single program in parallel while changing input data and parameters
  - ▶ Easy to describe these 3 points by the TFM API
    - ⊗ Set input data and parameters range
    - ⊗ Submit tasks
    - ⊗ Receive results



## Position of TFM API

- **TFM API targets on client-side programming**
  - ▶ A user don't have to care about remote side.
- **TFM API is implemented over the GridRPC framework to work on any GridRPC systems.**



## Users' requirements

- **Automatic task assignment to the machine**
  - ▶ Scheduling by performance and stability
    - ☞ Ex. Assignment priority, duplicated task submission
- **Fault-tolerant mechanism inside of the library**
  - ▶ Multiple retries until the task execution succeeds
  - ▶ Automatic recovery of the remote program
- **Simple API to program parameter generation and result analysis for TFM application**
  - ▶ Higher tools (Ex. TFM on Matlab) should be implemented for the specific application.
    - ☞ Ex. Interactive task execution, parameter generation
  - ▶ API for initializing a TFM environment



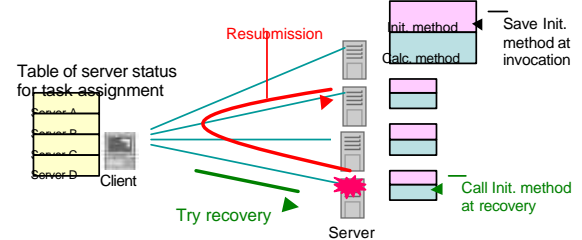
## Proposed TFM library

- **Support automatic task assignment**
  - ▶ Measured execution time reflects on the next assignment.
  - ▶ **Support automatic tuning of task window**
    - ☞ The window is tuned so that the total execution time will be minimum.
    - ☞ Users can specify MAX for limited memory capacity
  - ▶ If users want, they can specify the host by ID.
- **Support task completeness**
  - ▶ Multiple retries until the task execution succeeds
- **Support duplicated submission**
  - ▶ One of the two same tasks will succeed.
- **Support automatic recovery of the remote program**
  - ▶ Periodical check and recovery in background
- **Support automatic initialization of the remote program**
  - ▶ Initialization method is saved with data in the library for the recovery operation



## When a fault happens ...

- **A failed task is resubmitted to another host.**
- **A failed server is invoked and initialized by "Initialization method" saved in advance.**



## Proposed TFM API (1)

### ■ Initialization / finalization of TFM API library

```
int grpcg_init(char * conf, sched_attr_t * sched, ft_attr_t * ft);
int grpcg_fin();
```

### ■ Invoke a remote program (Ninf-G server)

```
int grpcg_remote_init(int num_pe, char * func, ...);
☞ All programs can have the same Initialization method.
int grpcg_remote_init_n(int server_id, int num_pe, char * func, ...);
☞ Each program can have a different initialization method with an ID.
```

### ■ Terminate a remote program

```
int grpcg_remote_fin(int num_pe);
int grpcg_remote_fin_n(int server_id, int num_pe);
```



## Proposed TFM API (2)

### ■ Task submission

```
int grpcg_submit(char * func, ...);
int grpcg_submit_n(int server_id, char * func, ...);
☞ Specify a target host of the task by server_id
int grpcg_submit_r(void * ref, char * func, ...);
☞ Set a pointer to the task for post-process
int grpcg_submit_nr(int server_id, void * ref, char * func, ...);
```

### ■ Wait for task completion

```
int grpcg_wait_all();
int grpcg_wait_any(int * task_id, void ** ref);
```

### ■ Task cancellation

```
int grpcg_cancel(int task_id);
```



# Sample program using TFM API

■ **Case: ED code of NAS Grid Benchmark**

```

:
rc = grpcg_init("server.list", &sched, NULL);
:
grpcg_remote_init(NUM_PE, NULL);
:
for(i=0; i<NUM_TASK; i++){
  grpcg_submit("SP.S", "SP", ..., &i, &width, &depth, ...);
}
rc = grpcg_wait_all();

grpcg_remote_fin(NUM_PE);
grpcg_fin();
:
    
```

Invoke a remote program without the initialization method

Submit a task without specifying the host (If a fault happens, the task will be resubmitted to anywhere else.)



# Implementation

■ **1. Prepare common components**

- ▶ Remote program (GridRPC server) management
- ▶ Task management
- ▶ Fault detection & background recovery of servers

■ **2. Implement TFM API**

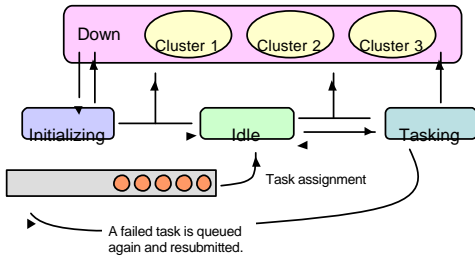
■ **Use of Middleware API of GridRPC and Ninf-G extensions**

- ▶ Argument Array (Argument Stack) API
- ▶ Remote object (Temporary storage function on remote)
- ▶ API to retrieve execution information of each RPC
- ▶ Complete non-blocking data transfer
- ▶ Invocation of multiple remote programs by one call



# Status mgmt. of task and server

- Servers are periodically sorted in the Idle pool.
- Down status is managed by each Handle Array.
  - ▶ Because recovery is operated by each Handle Array



# Use of Argument Array API

■ **Provided a new API (TFM API) over standard RPC calls**

- ▶ Ninf-G (Ver. 2.3) provides the Argument Stack API that will be redefined as the Argument Array API. The Argument Array API will be able to treat va\_list.

```

int grpcg_submit(char * func, ...){
:
va_start(ap, func);
grpc_arg_array_init_with_va_list(<handle>, arg_array, ap);
va_end(ap);
:
}

int grpcg_i_dispatch(){
:
grpc_call_arg_array_async(<handle>, &session_id, arg_array);
:
}
    
```

Submission

Execution

Handle is for checking arguments data type.

Create arg\_array from va\_list

Stored arguments are taken over.



# Feedback to GridRPC-WG

■ **Some extensions like that Ninf-G provides should be standardized in the GridRPC.**

- ▶ API to retrieve execution information on remote
  - ⊗ The information is useful for load balancing.
    - Ex. Transfer speed of arguments data, calculation cost
- ▶ Timing of data transfer
  - ⊗ Complete non-blocking transfer should be provided.
    - The TFM library can implement optimized transfer.

■ **"Arguments copy" function should be provided in the Argument Array API.**

- ▶ Reason 1: A user need to be careful not to rewrite the input data for the task.
- ▶ Reason 2: It is difficult to implement duplicate task submission.



# Summary

■ **Designed and implemented the Task Farming API library over the GridRPC**

- ▶ Based on the end-user API that is almost standardized
- ▶ Used the Argument Array API that is still being discussed
- ▶ Used the Ninf-G extensions that is not available in other GridRPC systems

■ **Revealed essential functionality to implement a higher-level API library such as TFM library**

- ▶ Some of them should be standardized in the GridRPC.
- ▶ Specially, the Argument Array API would be useful in many cases.

