Programming on the Grid using GridRPC

Yoshio Tanaka
(yoshio.tanaka@aist.go.jp)
Grid Technology Research Center, AIST
Outline

- **What is GridRPC?**
  - Overview
  - v.s. MPI
  - Typical scenarios

- **Overview of Ninf-G and GridRPC API**
  - Ninf-G: Overview and architecture
  - GridRPC API
  - Ninf-G API

- **How to develop Grid applications using Ninf-G**
  - Build remote libraries
  - Develop a client program
  - Run

- **Recent activities/achievements in Ninf project**
What is GridRPC?
Programming model on Grid based on Grid Remote Procedure Call (GridRPC)
Layered Programming Model/Method

**Portal / PSE**
- GridPort, HotPage,
- GPDK, Grid PSE Builder,
  etc…

**Easy but inflexible**

**High-level Grid Middleware**
- MPI (MPI CH-G2, PACX-MPI, …)
- GridRPC (Ninf-G, NetSolve, …)

**Low-level Grid Middleware**
- Globus Toolkit

**Difficult but flexible**

**Primitives**
- Socket, system calls, …
### Some Significant Grid Programming Models/Systems

- **Data Parallel**
  - MPI - MPICH-G2, GridMPI, PACX-MPI, …

- **Task Parallel**
  - GridRPC – Ninf, Netsolve, DIET, OmniRPC, …

- **Distributed Objects**
  - CORBA, Java/RMI, …

- **Data Intensive Processing**
  - DataCutter, Gfarm, …

- **Peer-To-Peer**
  - Various Research and Commercial Systems
    - UD, Entropia, JXTA, P3, …

- **Others…**
GridRPC

Utilization of remote Supercomputers

Call remote (special) libraries

Use as backend of portals / ASPs

Large-scale distributed computing using multiple computing resources on Grids

Suitable for implementing task-parallel applications (compute independent tasks on distributed resources)
GridRPC Model

**Client Component**
- Caller of GridRPC.
- Manages remote executables via function handles

**Remote Executables**
- Callee of GridRPC.
- Dynamically generated on remote servers.

**Information Manager**
- Manages and provides interface information for remote executables.
GridRPC: RPC “tailored” for the Grid

- Medium to Coarse-grained calls
  - Call Duration < 1 sec to > week

- Task-Parallel Programming on the Grid
  - Asynchronous calls, 1000s of scalable parallel calls

- Large Matrix Data & File Transfer
  - Call-by-reference, shared-memory matrix arguments

- Grid-level Security (e.g., Ninf-G with GSI)

- Simple Client-side Programming & Management
  - No client-side stub programming or IDL management

- Other features...
GridRPC v.s. MPI

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

* May be dynamic using process spawning
Typical scenario 1: desktop supercomputing

- Utilize remote supercomputers from your desktop computer
- Reduce cost for maintenance of libraries
- ASP-like approach

![Diagram Illustrating Desktop Supercomputing](image-url)
Typical scenario 2: parameter survey

- **Compute independent tasks on distributed resources**
  - e.g. combinatorial optimization problem solvers
- **Fault nodes can be discarded/retried**
- **Dynamic allocation / release of resources is possible**
Typical scenario 3: GridRPC + MPI

- Coarse-grained independent parallel (MPI) programs are executed on distributed clusters.
- Combine coarse-grained parallelism (by GridRPC) and fine-grained parallelism (by MPI).
- Dynamic migration of MPI jobs is possible.
- Practical approach for large-scale computation.
Sample Architecture and Protocol of GridRPC System – Ninf -

**Client side**
- **Call remote library**
  - Retrieve interface information
  - Invoke Remote Library Executable
  - It Calls back to the client

**Server side setup**
- Build Remote Library Executable
- Register it to the Ninf Server

**Generate**
- IDL file
- Numerical Library

**Remote Library Executable**

**IDL Compiler**

**Server side**
- Connect back

**Client**
- Interface Request
- Interface Reply
- Ninf Server
- Register
GridRPC: based on Client/Server model

- **Server-side setup**
  - Remote libraries must be installed in advance
    - Write IDL files to describe interface to the library
    - Build remote libraries
  - Syntax of IDL depends on GridRPC systems
    - e.g. Ninf-G and NetSolve have different IDL

- **Client-side setup**
  - Write a client program using GridRPC API
  - Write a client configuration file
  - Run the program
Ninf-G

Overview and Architecture
What is Ninf-G?

- A software package which supports programming and execution of Grid applications using GridRPC.
- The latest version is 2.3.0
- Ninf-G is developed using Globus C and Java APIs
  - Uses GSI, GRAM, MDS, GASS, and Globus-IO
- Ninf-G includes
  - C/C++, Java APIs, libraries for software development
  - IDL compiler for stub generation
  - Shell scripts to
    - compile client program
    - build and publish remote libraries
  - sample programs and manual documents
Globus Toolkit
Defacto standard as low-level Grid middleware
Requirements for Grid

- **Security**
  - authentication, authorization, message protection, etc.

- **Information services**
  - Provides various information
    - available resources (hw/sw), status, etc.

- **Resource management**
  - process spawning on remote computers

- **Scheduling**

- **Data management, data transfer**

- **Usability**
  - Single Sign On, etc.

- **Others**
  - accounting, etc...
What is the Globus Toolkit?

A Toolkit which makes it easier to develop computational Grids

Developed by the Globus Project Developer Team (ANL, USC/ISI)

Defacto standard as a low level Grid middleware

Most Grid testbeds are using the Globus Toolkit

Three versions are exist

- 2.4.3 (GT2 / Pre-WS)
- 3.2.1 (GT3 / OGSI)
- 3.9.4 (GT4 alpha / WSRF)

GT2 component is included in GT3/GT4

Pre-WS components
GT2 components

- **GSI**: Single Sign On + delegation
- **MDS**: Information Retrieval
  - Hierarchical Information Tree (GRIS+GIIS)
- **GRAM**: Remote process invocation
  - Three components:
    - Gatekeeper
    - Job Manager
    - Queuing System (pbs, sge, etc.)
- **Data Management**:
  - GridFTP
  - Replica management
  - GASS
- **Globus XIO**
- **GT2 provides C/Java APIs and Unix commands for these components**
GRAM: Grid Resource Allocation Manager

- Job request
- Job control, monitor
- Error notification

Gatekeeper

Job-manager

Queuing system
Big picture of the GT2

Diagram showing the process flow:
- **Client** connects to a **Proxy** system.
- **Proxy Cert.** is exchanged between client and proxy.
- The **gatekeeper** manages the grid-proxy-init process.
- Process invocation and data transfer from **Site B** to **Site C** through **GridFTP Server**.
- **Return result** is sent back to the client.
Some notes on the GT2 (1/2)

- **Globus Toolkit is not providing a framework for anonymous computing and mega-computing**
  - Users are required
    - to have an account on servers to which the user would be mapped when accessing the servers
    - to have a user certificate issued by a trusted CA
    - to be allowed by the administrator of the server
  - Complete differences with mega-computing framework such as SETI@HOME
Some notes on the GT2 (2/2)

Do not think that the Globus Toolkit solves all problems on the Grid.

- The Globus Toolkit is a set of tools for the easy development of computational Grids and middleware.
  - The Globus Toolkit includes low-level APIs and several UNIX commands.
  - It is not easy to develop application programs using Globus APIs. High-level middleware helps application development.

- Several necessary functions on the computational Grids are not supported by the Globus Toolkit.
  - Brokering, Co-scheduling, Fault Managements, etc.

- Other supposed problems
  - using IP-unreachable resources (private IP addresses + MPICH-G2)
  - scalability (ldap, maintenance of grid-mapfiles, etc.)
Ninf-G

Overview and architecture
Terminology

- **Ninf-G Client**
  - This is a program written by a user for the purpose of controlling the execution of computation.

- **Ninf-G IDL**
  - Ninf-G IDL (Interface Description Language) is a language for describing interfaces for functions and objects those are expected to be called by Ninf-G client.

- **Ninf-G Stub**
  - Ninf-G stub is a wrapper function of a remote function/object. It is generated by the stub generator according to the interface description for user-defined functions and methods.
Terminology (cont’d)

- **Ninf-G Executable**
  - Ninf-G executable is an executable file that will be invoked by Ninf-G systems. It is obtained by linking a user-written function with the stub code, Ninf-G and the Globus Toolkit libraries.

- **Session**
  - A session corresponds to an individual RPC and it is identified by a non-negative integer called Session ID.

- **GridRPC API**
  - Application Programming Interface for GridRPC. The GridRPC API is going to be standardized at the GGF GridRPC WG.
Sample Architecture and Protocol of GridRPC System – Ninf –

Client side

Server side

IDL file
Numerical Library

IDL Compiler

Generate

Remote Library Executable

fork

Register

Ninf Server

Invoke Executable

Connect back

Interface Reply

Interface Request

Client

Client side

Server side

fork
How to use Ninf-G

- **Build remote libraries on server machines**
  - Write IDL files
  - Compile the IDL files
  - Build and install remote executables

- **Develop a client program**
  - Programming using GridRPC API
  - Compile

- **Run**
  - Create a client configuration file
  - Generate a proxy certificate
  - Run
Sample Program

Parameter Survey

- No. of surveys: n
- Survey function: survey(in1, in2, result)
- Input Parameters: double in1, int in2
- Output Value: double result[]

Main Program

```c
int main(int argc, char** argv) {
    int i, n, in2;
    double in1, result[100][100];
    Pre_processing();
    for (i = 0; i < n; i++){
        survey(in1, in2, result+100*n)
    }
    Post_processing();
}
```

Survey Function

```c
survey(double in1, int in2, double* result) {
    ...
    Do Survey
    ...
```
Build remote library (server-side operation)

Original Program

Client Program

Callee Function

IDL File

Module Survey_prog;

Define survey
(IN double in1, IN int in2, IN int size, OUT double* result);

Required "survey.o"
Calls "C" survey(in1, in2, size, result);

survey (double in1, int in2, int size, double* result)

Do Survey

Specify size of argument

Callee Function
Int main(int argc, char** argv) {
    int i, n, in2;
    double in1, result[100][100];
    Pre_processing();
    for(I = 0; I < n; i++) {
        handle[i] = grpc_function_handle_init();
    }
    for(I = 0; I < n, i++) {
        grpc_call_async(handles, in1, in2, 100, result + 100 * n)
    }
    grpc_wait_all();
    for(I = 0; i < n; i++) {
        grpc_function_handle_destruct();
    }
    grpc_finalize();
    Post_processing();
}
Ninf-G

How to build remote libraries
Ninf-G remote libraries

- Ninf-G remote libraries are implemented as executable programs (Ninf-G executables) which contain stub routine and the main routine.
- The stub routine will be spawned off by GRAM.
- The stub routine handles communication with clients and Ninf-G system itself.
- Argument marshalling.
- Underlying executable (main routine) can be written in C, C++, Fortran, etc.
Ninf-G remote libraries (cont’d)

- **Ninf-G provides two kinds of Ninf-G remote executables:**
  - **Function**
    - Stateless
    - Defined in standard GridRPC API
  - **Ninf-G object**
    - stateful
    - enables to avoid redundant data transfers
    - multiple methods can be defined
      - initialization
      - computation
How to build Ninf-G remote libraries (1/3)

Write an interface information using Ninf-G Interface Description Language (Ninf-G IDL).
Example:

Module mmul;
Define dmmul (IN int n,
  IN double A[n][n],
  IN double B[n][n],
  OUT double C[n][n])
Require “libmmul.o”
Calls “C” dmmul(n, A, B, C);

Compile the Ninf-G IDL with Ninf-G IDL compiler

% ng_gen <IDL_FILE>

ns_gen generates stub source files and a makefile (<module_name>.mak)
How to build Ninf-G remote libraries (2/3)

Compile stub source files and generate Ninf-G executables and LDIF files (used to register Ninf-G remote libs information to GRIS).

```
% make -f <module_name>.mak
```

Publish the Ninf-G remote libraries

```
% make -f <module_name>.mak install
```

This copies the LDIF files to

```
${GLOBUS_LOCATION}/var/gridrpc
```
How to build Ninf-G remote libraries (3/3)

- **Ninf-G IDL file**
  - `<module>.idl`

- **ns_gen**

- **GRI S**

- GRAM

- **<module>::foo.ldif**
- **<module>::bar.ldif**
- **<module>::goo.ldif**

- **Library program**
  - `libfoo.a`

- **<module>::mak**

- **make -f <module>::mak**

- **stub_foo.c**
- **stub_bar.c**
- **stub_goo.c**
Ninf-G IDL Statements (1/3)

- **Module** `module_name`
  - specifies the module name.

- **CompileOptions** "options"
  - specifies compile options which should be used in the resulting makefile.

- **Library** "object files and libraries"
  - specifies object files and libraries.

- **FortranFormat** "format"
  - provides translation format from C to Fortran.
  - Following two specifiers can be used:
    - `%s`: original function name
    - `%l`: capitalized original function name
  - Example:
    ```
    FortranFormat "%l_";
    Calls "Fortran" fft(n, x, y);
    will generate function call
    _FFT_(n, x, y);
    in C.
    ```

- **Globals** `{ ... C descriptions }`
  - declares global variables shared by all functions
How to define a remote function

Define `routine_name(parameters...)`

- `[“description”]`
- `[Required “object files or libraries”]`
- `[Backend “MPI”|”BLACS”]`
- `[Shrink “yes”|”no”]`
- `{C descriptions} | Calls “C”|”Fortran” calling sequence}`

- declares function interface, required libraries and the main routine.

- Syntax of parameter description:
  - `[mode-spec] [type-spec] formal_parameter
  - `[dimension [:range]]+]+`
How to define a remote object

```
DefClass class name

["description"]
[Required "object files or libraries"]
[Backend "MPI" | "BLACS"]
[Language "C" | "fortran"]
[Shrink "yes" | "no"]
{ [DefState{ ... }]
  DefMethod method name (args...)
  {calling sequence}

▶ Declares an interface for Ninf-G objects
```
Syntax of parameter description (detailed)

- **mode-spec:** one of the following
  - IN: parameter will be transferred from client to server
  - OUT: parameter will be transferred from server to client
  - INOUT: at the beginning of RPC, parameter will be transferred from client to server. at the end of RPC, parameter will be transferred from server to client
  - WORK: no transfers will be occurred. Specified memory will be allocated at the server side.

- **type-spec** should be either `char, short, int, float, long, longlong, double, complex, or filename`.

For arrays, you can specify the size of the array. The size can be specified using scalar IN parameters.

- Example: IN int n, IN double a[n]
Matrix Multiply

Module matrix;

Define dmmul (IN int n,
    IN double A[n][n],
    IN double B[n][n],
    OUT double C[n][n])

"Matrix multiply: C = A x B"

Required "libmmul.o"

Calls "C" dmmul(n, A, B, C);
Module sample_object;

DefClass sample_object
"This is test object"
Required "sample.o"
{
    DefMethod mmul(IN long n, IN double A[n][n],
                   IN double B[n][n], OUT double C[n][n])
    Calls "C" mmul(n,A,B,C);

    DefMethod mmul2(IN long n, IN double A[n*n+1-1],
                   IN double B[n*n+2-3+1], OUT double C[n*n])
    Calls "C" mmul(n,A,B,C);

    DefMethod FFT(IN int n, IN int m, OUT float x[n][m], float INOUT y[m][n])
    Calls "Fortran" FFT(n,x,y);
}
Sample Ninf-G IDL (3/3)

**ScaLAPACK (pdgesv)**

```c
Module SCALAPACK;

CompileOptions "NS_COMPILER = cc";
CompileOptions "NS_LINKER = f77";
CompileOptions "CFLAGS = -DAdd_ -O2 -64 -mips4 -r10000";
CompileOptions "FFLAGS = -O2 -64 -mips4 -r10000";
Library "scalapack.a pblas.a redista tools.a libmpiblacs.a -lblas -lmpi -lm";

Define pdgesv (IN int n, IN int nrhs, INOUT double global_a[n][lda:n], IN int lda,
            INOUT double global_b[nrhs][ldb:n], IN int ldb, OUT int info[1])

Backend "BLACS"
Shrink "yes"
Required "procmap.o pdgesv_ninf.o ninf_make_grid.of Cnumroc.o descinit.o"
Calls "C" ninf_pdgesv(n, nrhs, global_a, lda, global_b, ldb, info);
```
Ninf-G

How to call Remote Libraries
- client side APIs and operations -
(Client) User’s Scenario

- Write client programs in C/C++/Java using APIs provided by Ninf-G
- Compile and link with the supplied Ninf-G client compile driver (ngcc)
- Write a client configuration file in which runtime environments can be described
- Run `grid-proxy-init` command
- Run the program
GridRPC API / Ninf-G API
APIs for programming client applications
The GridRPC API and Ninf-G API

GridRPC API
- Standard C API defined by the GGF GridRPC WG.
- Provides portable and simple programming interface.
- Enable interoperability between implementations such as Ninf-G and NetSolve.

Ninf-G API
- Non-standard API (Ninf-G specific)
- Complement to the GridRPC API
- Provided for high performance, usability, etc.
- Ended by _np
  
  eg: grpc_function_handle_array_init_np(…)

[Logos and icons]
Rough steps for RPC

**Initialization**

```c
grpc_initialize(config_file);
```

**Create a function handle**

- Abstraction of a connection to a remote executable

```c
grpc_function_handle_t handle;
grpc_function_handle_init(&handle, host, port, "lib_name");
```

**Call a remote library**

```c
grpc_call(&handle, args...);
or
grpc_call_async(&handle, args...);
grpc_wait();
```
Data types

- **Function handle** – `grpc_function_handle_t`
  - A structure that contains a mapping between a client and an instance of a remote function

- **Object handle** – `grpc_object_handle_t_np`
  - A structure that contains a mapping between a client and an instance of a remote object

- **Session ID** – `grpc_sessionid_t`
  - Non-negative integer that identifies a session
  - Session ID can be used for status check, cancellation, etc. of outstanding RPCs.

- **Error and status code** – `grpc_error_t`
  - Integer that describes error and status of GridRPC APIs.
  - All GridRPC APIs return error code or status code.
Initialization / Finalization

- **grpc_error_t grpc_initialize(char *config_file_name)**
  - Reads the configuration file and initialize client.
  - Any calls of other GRPC APIs prior to grpc_initialize would fail.
  - Returns GRPC_OK (success) or GRPC_ERROR (failure)

- **grpc_error_t grpc_finalize()**
  - Frees resources (memory, etc.).
  - Any calls of other GRPC APIs after grpc_finalize would fail.
  - Returns GRPC_OK (success) or GRPC_ERROR (failure)
Function handles

```
grpc_error_t grpc_function_handle_default(
    grpc_function_handle_t *handle,
    char *func_name)
```

- Creates a function handle to the default server

```
grpc_error_t grpc_function_handle_init(
    grpc_function_handle_t *handle,
    char *host_port_str,
    char *func_name)
```

- Specifies the server explicitly by the second argument.

```
grpc_error_t grpc_function_handle_destruct(
    grpc_function_handle_t *handle)
```

- Frees memory allocated to the function handle
Function handles (cont’d)

- `grpc_error_t grpc_function_handle_array_default_np (grpc_function_handle_t *handle, size_t nhandles, char *func_name)`
  - Creates multiple function handles via a single GRAM call.

- `grpc_error_t grpc_function_handle_array_init_np (grpc_function_handle_t *handle, size_t nhandles, char *host_port_str, char *func_name)`
  - Specifies the server explicitly by the second argument.

- `grpc_error_t grpc_function_handle_array_destruct_np (grpc_function_handle_t *handle, size_t nhandles)`
  - Specifies the server explicitly by the second argument.
Object handles

- **grpc_error_t grpc_object_handle_default_np (.grpc_object_handle_t_np *handle, char *class_name)**
  - Creates an object handle to the default server.

- **grpc_error_t grpc_object_handle_init_np (grpc_function_object_t_np *handle, char *host_port_str, char *class_name)**
  - Specifies the server explicitly by the second argument.

- **grpc_error_t grpc_function_object_destruct_np (grpc_object_handle_t_np *handle)**
  - Frees memory allocated to the function handle.
Object handles (cont’d)

- `grpc_error_t grpc_object_handle_array_default (grpc_object_handle_t_np *handle, size_t nhandles, char *class_name)`
  - Creates multiple object handles via a single GRAM call.

- `grpc_error_t grpc_object_handle_array_init_np (grpc_object_handle_t_np *handle, size_t nhandles, char *host_port_str, char *class_name)`
  - Specifies the server explicitly by the second argument.

- `grpc_error_t grpc_object_handle_array_destruct_np (grpc_object_handle_t_np *handle, size_t nhandles)`
  - Frees memory allocated to the function handles.
Synchronous RPC v.s. Asynchronous RPC

**Synchronous RPC**
- Blocking Call
- Same semantics with a local function call.

```
grpc_call(...);
```

**Asynchronous RPC**
- Non-blocking Call
- Useful for task-parallel applications

```
grpc_call_async(...);
grpc_wait_*(...);
```
RPC functions

- `grpc_error_t grpc_call (grpc_function_handle_t *handle, ...)`
  - Synchronous (blocking) call

- `grpc_error_t grpc_call_async (grpc_function_handle_t *handle, grpc_sessionid_t *sessionID, ...)`
  - Asynchronous (non-blocking) call
  - Session ID is stored in the second argument.
Ninf-G method invocation

- `grpc_error_t grpc_invoke_np (grpc_object_handle_t_np *handle, char *method_name, ...)
  - Synchronous (blocking) method invocation

- `grpc_error_t grpc_invoke_async_np (grpc_object_handle_t_np *handle, char *method_name, grpc_sessionid_t *sessionID, ...)
  - Asynchronous (non-blocking) method invocation
  - session ID is stored in the third argument.
Session control functions

- **grpc_error_t grpc_probe (grpc_sessionid_t sessionID)**
  - probes the job specified by SessionID whether the job has been completed.

- **grpc_error_t grpc_probe_or (grpc_sessionid_t *idArray, size_t length, grpc_sessionid_t *idPtr)**
  - probes whether at least one of jobs in the array has been

- **grpc_error_t grpc_cancel (grpc_sessionid_t sessionID)**
  - Cancels a session

- **grpc_error_t grpc_cancel_all ()**
  - Cancels all outstanding sessions
Wait functions

- `grpc_error_t grpc_wait (grpc_sessionid_t sessionID)`
  - Waits outstanding RPC specified by `sessionID`

- `grpc_error_t grpc_wait_and (grpc_sessionid_t *idArray, size_t length)`
  - Waits all outstanding RPCs specified by an array of `sessionIDs`
Wait functions (cont’d)

```c
grpc_error_t grpc_wait_or (  
grpc_sessionid_t *idArray,  
size_t length,  
grpc_sessionid_t *idPtr)
```

- Waits any one of RPCs specified by an array of sessionIDs.

```c
grpc_error_t grpc_wait_all ()
```

- Waits until all outstanding RPCs are completed.

```c
grpc_error_t grpc_wait_any (  
grpc_sessionid_t *idPtr)
```

- Waits any one of outstanding RPCs.
Ninf-G

Compile and run
Prerequisite

Environment variables
- GPT_LOCATION
- GLOBUS_LOCATION
- NG_DIR

PATH
- ${GLOBUS_LOCATION}/etc/globus-user-env.{csh,sh}
- ${NG_DIR}/etc/ninfg-user-env.{csh,sh}

Globus-level settings
- User certificate, CA certificate, grid-mapfile
- test
  % grid-proxy-init
  % globus-job-run server.foo.org /bin/hostname

Notes for dynamic linkage of the Globus shared libraries:
- Globus dynamic libraries (shared libraries) must be linked with the Ninf-G stub executables. For example on Linux, this is enabled by adding ${GLOBUS_LOCATION}/lib in /etc/ld.so.conf and run ldconfig command.
Compile and run

- **Compile the client application using** `ngcc` **command**
  \[
  \% \text{ngcc} \ -o \text{myapp} \text{app.c}
  \]

- **Create a proxy certificate**
  \[
  \% \text{grid-proxy-init}
  \]

- **Prepare a client configuration file**

- **Run**
  \[
  \% \text{./myapp config.cl} \ [\text{args...}]
  \]
Client configuration file

- Specifies runtime environments
- Available attributes are categorized to sections:
  - INCLUDE section
  - CLIENT section
  - LOCAL_LDIF section
  - FUNCTION_INFO section
  - MDS_SERVER section
  - SERVER section
  - SERVER_DEFAULT section
Frequently used attributes

- `<CLIENT> </CLIENT> section`
  - loglevel
  - refresh_credential

- `<SERVER> </SERVER> section`
  - hostname
  - mpi_runNoOfCPUs
  - jobmanager
  - job_startTimeout
  - job_queue
  - heartbeat / heartbeat_timeoutCount
  - redirect_outerr

- `<FUNCTION_INFO> </FUNCTION_INFO> section`
  - session_timeout

- `<LOCAL_LDIF> </LOCAL_LDIF> section`
  - filename
Example: Task Parallel Programs
(Compute PI using Monte-Carlo Method)

Generate a large number of random points within the square region that exactly encloses a unit circle (1/4 of a circle)

\[ \pi = 4(p) \]
Module pi;

Define pi_trial (  
    IN int seed,  
    IN long times,  
    OUT long * count)  
"monte carlo pi computation"  
Required "pi_trial.o"  
{
    long counter;
    counter = pi_trial(seed, times);
    *count = counter;
}

long pi_trial (int seed, long times)  
{
    long l, counter = 0;

    srandom(seed);
    for (l = 0; l < times; l++)  
    {
        double x =  
            (double)random() / RAND_MAX;
        double y =  
            (double)random() / RAND_MAX;

        if (x * x + y * y < 1.0)  
            counter++;
    }
    return counter;
}
Compute PI - Client Side-

```c
#include "grpc.h"
#define NUM_HOSTS 8
char * hosts[] =
   {"host00", "host01", "host02", "host03",
    "host04", "host05", "host06", "host07"}

main(int argc, char ** argv){
    double pi;
    long times, count[NUM_HOSTS], sum;
    char * config_file;
    int i;
    if (argc < 3){
        fprintf(stderr,
            "USAGE: %s CONFIG_FILE TIMES \n",
            argv[0]);
        exit(2);
    }
    config_file = argv[1];
    times = atol(argv[2]) / NUM_HOSTS;
    /* Initialize */
    if (grpc_initialize(config_file)
        != GRPC_OK){
        grpc_perror("grpc_initialize");
        exit(2);
    }
    /* Initialize Function Handles */
    for (i = 0; i < NUM_HOSTS; i++)
        grpc_function_handle_init(&handles[i],
            hosts[i], port, "pi/pi_trial");
    for (i = 0; i < NUM_HOSTS; i++)
        /* Asynchronous RPC */
        if (gprc_call_async(&handles[i], i,
            times, &count[i]) ==
            GRPC_ERROR){
            grpc_perror("pi_trial");
            exit(2);
        }
    /* Wait all outstanding RPCs */
    if (grpc_wait_all() == GRPC_ERROR){
        grpc_perror("wait_all");
        exit(2);
    }
    /* Display result */
    for (i = 0, sum = 0; i < NUM_HOSTS; i++)
        sum += count[i];
    pi = 4.0 *
        ( sum / ((double) times * NUM_HOSTS));
    printf("PI = %f\n", pi);
    /* Finalize */
    grpc_finalize();
}
```
Ninf-G

Summary
How to use Ninf-G (again)

- **Build remote libraries on server machines**
  - Write IDL files
  - Compile the IDL files
  - Build and install remote executables

- **Develop a client program**
  - Programming using GridRPC API
  - Compile

- **Run**
  - Create a client configuration file
  - Generate a proxy certificate
  - Run
Ninf-G tips

How the server can be specified?
- Server is determined when the function handle is initialized.
  - `grpc_function_handle_init();`
    - hostname is given as the second argument
  - `grpc_function_handle_default();`
    - hostname is specified in the client configuration file which must be passed as the first argument of the client program.
- Ninf-G does not provide broker/scheduler/meta-server.

Should use LOCAL LDIF rather than MDS.
- easy, efficient and stable

How should I deploy Ninf-G executables?
- Deploy Ninf-G executables manually
- Ninf-G provides automatic staging of executables

Other functionalities?
- heatbeating
- timeout
- client callbacks
- attaching to debugger
- …
Ninf-G

Recent achievements
Climate simulation on AIST-TeraGrid @SC2003

Ninf-G

Client (AIST)

Severs
NCSA Cluster (225 CPU)
Experiments on long-run

**Purpose**
- Evaluate quality of Ninf-G2
- Have experiences on how GridRPC can adapt to faults

**Ninf-G stability**
- Number of executions: 43
- Execution time
  - (Total): 50.4 days
  - (Max): 6.8 days
  - (Ave): 1.2 days
- Number of RPCs: more than 2,500,000
- Number of RPC failures: more than 1,600
  - (Error rate is about 0.064 %)

![Graph showing number of alive servers over elapsed time]
Re-implementation using GridRPC

Original implementation (MPI)

New implementation (GridRPC + MPI)
Hybrid QM-MD Simulation of Nano-structured Si in Corrosive Environment

Nano-structured Si system under stress

- two slabs connected with a slanted pillar
- 0.11 million atoms

4 quantum regions:

#0: 69 atoms including $2\text{H}_2\text{O} + 2\text{OH}$

#1: 68 atoms including $\text{H}_2\text{O}$

#2: 44 atoms including $\text{H}_2\text{O}$

#3: 56 atoms including $\text{H}_2\text{O}$

Close-up view
Testbed used in the experiment

P32 (512 CPU)

#0: 69 atoms
including $2\text{H}_2\text{O}+2\text{OH}$

P32 (512 CPU)

#1: 68 atoms
including $\text{H}_2\text{O}$

F32 (256 CPU)

#2: 44 atoms
including $\text{H}_2\text{O}$

TCS (512 CPU) @ PSC

#3: 56 atoms
including $\text{H}_2\text{O}$
QM/MD simulation over the Pacific

Initial set-up

Calculate MD forces of QM+MD regions

Calculate MD forces of QM region

Update atomic positions and velocities

Calculate QM force of the QM region

Total number of CPUs: 1792

P32 (512 CPU)

P32 (512 CPU)

F32 (256 CPU)

TCS (512 CPU)

QM Server

MD Client

Ninf-G

QM Server

@ PSC
• Total number of CPUs: 1792
• Total Simulation Time: 10 hour 20 min
• # steps: 10 (= 7fs)
• Average time / step: 1 hour
• Size of generated files / step: 4.5GB
Ninf-G3 and Ninf-G4

- **Ninf-G3**: based on GT3
- **Ninf-G4**: based on GT4
- Ninf-G3 and Ninf-G4 invoke remote executables via WS GRAM.
- **Ninf-G3 alpha was released in Nov. 2003.**
  - GT 3.2.1 was so immature that Ninf-G3 is not practical for use 😞
- **We are now tackling with GT4 😊**
  - GT 3.9.4 is still alpha version and it does not provide C client API.
  - Ninf-G4 alpha that supports Java client is ready for evaluation of GT4.
For more info, related links

- Ninf project ML
  - ninf@apgrid.org

- Ninf-G Users’ ML
  - ninf-users@apgrid.org

- Ninf project home page
  - http://ninf.apgrid.org

- Global Grid Forum
  - http://www.ggf.org/

- GGF GridRPC WG

- Globus Alliance
  - http://www.globus.org/