Design, implementation and performance evaluation of GridRPC programming middleware for a large-scale computational Grid

Yoshio Tanaka, Hiroshi Takemiya, Hidemoto Nakada, Satoshi Sekiguchi
Grid Technology Research Center, AIST
Tsukuba Central 2, Umezono 1-1-1, Tsukuba, Ibaraki 305-8568, Japan
{yoshio.tanaka,h-takemiya,hide-nakada,s.sekiguchi}@aist.go.jp

Abstract

This paper reports on the design, implementation and performance evaluation of a suite of GridRPC programming middleware called Ninf-G Version 2 (Ninf-G2). Ninf-G2 is a reference implementation of the GridRPC API, a proposed GGF standard. Ninf-G2 has been designed so that it provides 1) high performance in a large-scale computational Grid, 2) the rich functionalities which are required to adapt to compensate for the heterogeneity and unreliability of a Grid environment, and 3) an API which supports easy development and execution of Grid applications. Ninf-G2 is implemented to work with basic Grid services, such as GSI, GRAM, and MDS in the Globus Toolkit version 2. The performance of Ninf-G2 was evaluated using a weather forecasting system which was developed using Ninf-G2. The experimental results indicate that high performance can be attained even in relatively fine-grained task-parallel applications on hundreds of processors in a Grid environment.

1. Introduction

GridRPC is a programming model based on a Remote Procedure Call (RPC) mechanism tailored for the Grid. Although when viewed at a very high level, the programming model provided by GridRPC is that of standard RPC plus asynchronous, coarse-grained parallel tasking, in practice there are a variety of features that will largely hide the dynamic, insecure, and unstable aspects of the Grid from programmers. Some representative GridRPC systems are Netsolve [1], Ninf [2, 3], Ninf-G [4], DIET [5], and OmniRPC [6]. Providing simple, yet powerful, client-server-based frameworks for programming on the Grid, those systems have seen successful usage in various Grid application projects such as a general monte carlo simulator of cellular micro-physiology [7], short- to middle-term weather forecasting on an international Grid testbed [8], and an optimization problem solver using a parallel branch and bound algorithm in Grid environments [9]. Although those efforts reported performance evaluation and feasibility studies of GridRPC, most experiments had been made on small-scale Grid environments. Our previous studies reported that Grid middleware such as the Globus Toolkit [10] and Ninf-G have problems with performance and functionalities for utilizing a large-scale computational Grid [8].

This paper reports on the design, implementation, and performance evaluation of a GridRPC programming middleware suite called Ninf-G Version 2 (Ninf-G2). Ninf-G2 is a reference implementation of the GridRPC API, a proposed GGF standard. Ninf-G2 aims to support development and execution of Grid applications which will run efficiently on a large-scale computational Grid. Here, we use the term "large-scale computational Grid" as a cluster of about ten geographically distributed cluster systems, each of which consists of tens to hundreds of processors.

The next section introduces an overview of the GridRPC API. Section 3 describes the design and implementation of Ninf-G2. Preliminary performance evaluation using a weather forecasting system which was developed using Ninf-G2 is shown in Sections 4 and 5. A conclusion, and plans for future work are described in Section 6.

2. Overview of GridRPC

A GridRPC system generally consists of the following four components; **Client Component**, **Server Component**, **Remote Executable**, and **Information Service**. Client Components are programs that issue requests for GridRPC invocations. Each component consists of a user's main program and the GridRPC library. A Server Component invokes Remote Executables as described below. Remote Executables perform the actual computation at the Server. Each component consists of a user supplied server-side compute routine, a system generated stub main program, and a system-supplied communication library. Information service provides various information for the client component to invoke and to use to communicate with the Remote Executable component.

Two fundamental objects in the GridRPC model are *function handles* and *session IDs*. The function handle represents a mapping from a function name to an instance of that function on a particular server. The GridRPC API does not dictate the mechanics of resource discovery since different underlying GridRPC implementations may use vastly different protocols. Once a particular function-to-server mapping has been established by initializing a function handle, all RPC calls using that function handle will be executed on the server specified in that binding. A session ID is an identifier representing a particular non-blocking RPC call. The session ID is used throughout the API to allow users to obtain the status of a previously submitted non-blocking call, to wait for a call to complete, to cancel a call, or to check the error code of a call.

3. Design and Implementation

This section describes issues which were considered when Ninf-G2 was designed and details of the implementation of Ninf-G2.

3.1. Design Policy

Ninf-G2 has been designed so that it provides 1) high performance in a large-scale computational Grid, 2) rich functionalities which are required to compensate for the heterogeneity and unreliability of a Grid environment, and 3) an API which supports easy development and execution of Grid applications. The following is a detailed description of those issues.

- **Reducing Overhead for Initialization of Function Handles**
  In order to utilize hundreds to thousands of processors efficiently, it is necessary to reduce overhead for initialization of function handles to a level as low as possible. This overhead includes information retrieval which is necessary for calling remote executables, authentication and authorization, and launching of Remote Executables etc.
- **Making Data Transfers Efficient**
  Data transfer between a client and a server is one of the most severe problem from performance point of view. It is necessary to design light-weight protocols for data transfers and to provide functionalities which reduce overhead for data transfers. The overhead includes redundant data transfers between a client and servers.
- **Compensating for the Heterogeneity and Unreliability of a Grid Environment**
  In Grid environments, it is usual that the configuration of servers differs greatly from one to another. For example, local queuing systems available, the port number on which a server daemon is listening, protocols available for establishing connections, paths of executable programs, and default environment variables are server-specific attributes. Unreliability is another characteristics of Grid environments. Resource usage on Grid environments is highly dynamic. Any resource may suddenly become heavily loaded and the network or the resource itself may go down during the execution of applications. Grid programming middleware must be capable of adapting to this heterogeneity and unreliability.
- **Supporting the Debugging of Applications**
  The GridRPC API provides simple, yet powerful, client-server-based frameworks for programming on the Grid. However, in order to support the debugging of applications, it is necessary to provide APIs and functionalities for this debugging.
- **Keeping the system as compact as possible**
  The GridRPC API specifies a primitive API for GridRPC and it does not include additional features such as scheduling and fault tolerance. Ninf-G2 is designed focusing on simplicity. Ninf-G2 does not provide fault recovery, or load-balancing by itself. Ninf-G2 propagates an appropriate error status to the client if the error occurs at either the client or the server machines, however Ninf-G2 itself does not try to recover from the errors.

3.2. Overview of the Implementation

Ninf-G2 is implemented to work with basic Grid services, such as GSI, GRAM, and MDS in the Globus Toolkit version 2. Ninf-G2 employs the following components from the Globus Toolkit, as shown in Figure 1. **GRAM (Globus Resource Allocation Manager** invokes remote executables. **MDS (Monitoring and Discovery Service)** publishes interface information and pathnames of GridRPC components. **Globus-IO** is used for the communication between clients and remote executables. **GASS** redirects stdout and stderr of the GridRPC component to the client tty.

At the server side, Ninf-G IDL (Interface Description Language) is used to describe interface information for remote executables. The Ninf-G IDL compiler generates a makefile and interface information source files from which the interface information files in XML, as well as in the LDIF (LDAP Data Interchange Format) will be generated by running the "make" command. The information files include interface information (in XML), the pathname of the remote executable, and the identifier of the remote executable. The LDIF file is used to push the information to the MDS server.

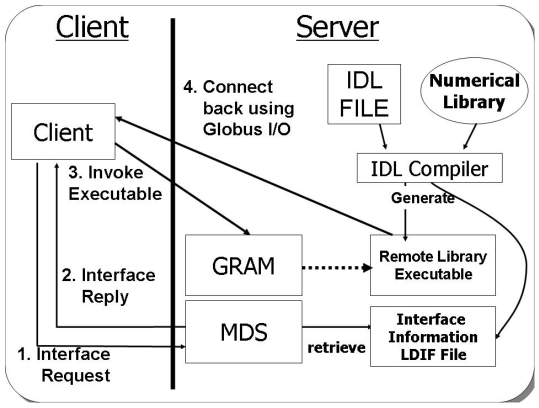

Figure 1. Overview of Ninf-G.

In order to call remote executables, the client is required to carry out the three steps; (1) Retrieval of interface information and executable pathname, (2) Invocation of the remote executable, and (3) Acceptance of a connection from the remote executable. Each of these steps becomes overhead for initialization of applications, and this overhead is non-negligible when many function handles are launched on a large-scale cluster. In order to implement light-weight initialization of functions handles, it is necessary to provide solutions for reducing the overhead.

3.3. Design and Implementation

In designing and implementing Ninf-G2, we considered the issues described in Section 3.1.

3.3.1. Solutions for Reducing Overhead for Initialization of Function Handles

- **Create multiple function handles via a single GRAM call and provide an API for utilizing the functionality.**
  A single GRAM call usually takes several seconds for GSI authentication and a process invocation via the Globus jobmanager. This indicates that it will take more than several minutes to tens minutes for hundreds of GRAM calls on a large-scale cluster. Also, many Globus jobmanager processes which will be launched on the front-end node will increase the load on the front-end node and cause the creation of additional overhead.

  Ninf-G2 implements a functionality which enables the creation of multiple function handles via a single GRAM call and provides an API for utilizing the functionality. For example, *grpc_function_handle_array_default_np()* takes three arguments, a pointer to an array of function handles, the number of function handles, and the name of the remote executable. When *grpc_function_handle_array_default_np()* is invoked, Ninf-G2 will construct an RSL in which the *count* attribute is specified as the number of function handles, and pass the RSL to the Globus GRAM. This allows invocation of multiple remote executables, i.e. initialization of multiple function handles, via a single GRAM call.

- **Bypass MDS lookup for information retrieval**
  Querying an MDS server for getting information on remote executables is a more difficult problem from a performance point of view, since it takes several minutes if the MDS server contains a large MDS tree. Although a useful resource discovery mechanism is essential for the acceptance of grid computing, we need to provide a practical scheme for information retrieval. Several approaches could be candidates for the implementation of information retrieval. For example in CORBA, both a client and servers generate stubs and share information statically. Although this approach is straightforward and reduces the overhead for information retrieval, client programmers need to prepare IDL files for stub generation that constitutes a burden on client programmers. Ninf-G2 implements a functionality which enables it to retrieve the necessary information not from an MDS server, but from LDIF files which are placed on the client machine in advance. When Ninf-G executables are generated on the server machine, the LDIF files are generated by the Ninf-G IDL compiler as well. The LDIF files should be copied to the client machine and can be specified in the client configuration file which is passed to the application as the first argument.

  The use of client-side LDIF files is practical, however the approach can not retrieve up-to-date information dynamically from the information server. Resource discovery mechanism is an important feature of Grid and we expect that the index services in the next version of the Globus Toolkit provides enough stability, scalability and high performance to be used in information retrieval in Ninf-G.

3.3.2. Solutions for Making Data Transfers Efficient

Ninf-G2 provides the following three new functionalities for efficient data transfers and elimination of redundant data transfers.

- **Implementation of Ninf-G remote object**
  Although the semantics of a remote executable is “stateless,” it is desirable to provide a “stateful” remote executable since typical applications repeat computation for large data sets with different parameters. In the case of “stateless” executables, it needs to send the data in every remote li-

brary call, which would be a severe problem in a Grid environment. Ninf-G2 provides a "stateful" remote executable as a "Ninf-G remote object." A Ninf-G remote object can hold a "state" and be used to eliminate redundant data transfers between a client and servers. Ninf-G2 provides API functions such as *grpc_object_handle_init_np()* and *grpc_invoke_np()* for utilizing Ninf-G remote objects. *grpc_object_handle_init_np()* initializes a Ninf-G remote object and creates an object handle which is an represents a connection between the client and the Ninf-G remote object. *grpc_invoke_np()* calls methods of the Ninf-G remote object as described in the Ninf-G IDL. Ninf-G remote object is an instance of a class which is defined in an IDL file using *DefClass* statement in the server side. Multiple methods, which can be invoked by a client using the client API such as *grpc_invoke_np()*, can be defined in a class using *DefMethod* statement.

- **Binary protocol for data transfers**
  Ninf-G2 enables the use of the Binary protocol as well as the XML protocol for data transfers between a client and a server. The protocol can be specified in the client configuration file.
- **Compression of transfered data**
  Ninf-G2 enables data transfers with compression. A flag which specifies whether to enable or disable data compression, and a data size as the threshold for compressing data can be specified in the client configuration file.

3.3.3. Solutions for Compensating for the Heterogeneity and Unreliability of a Grid Environment In order to compensate for the heterogeneity and unreliability of a Grid environment, Ninf-G2 provides the following five functionalities:

- **Client configuration formats for detailed description of server attributes**
  The GridRPC API specifies that the first argument of a client program must be a "client configuration file" in which information required for running applications is described. In order to compensate for the heterogeneity and unreliability of a Grid environment, Ninf-G2 provides client configuration formats for detailed description of server attributes such as the Globus jobmanager and a protocol for data transfers, etc.
- **Timeout value for initialization of a function handle and RPC**
  If a server machine is fully utilized, requests for initialization of function handles and remote library calls may be stuck in the queue and will not be launched for a long time, and this may cause deadlock of applications. Ninf-G2 provides a functionality to specify a timeout value for initialization of function handles as well as remote library calls. The timeout values can be specified in the client configuration file.
- **Heartbeat**
  Remote executable reports a heartbeat message to the client at a pre-specified interval. Ninf-G2 provides an API function for checking the heartbeat from the remote executable. The interval can be specified in the client configuration file.
- **Client Callbacks**
  Ninf-G2 provides a functionality called "client callbacks" by which a remote executable calls a function on the client machine. The client callback can be used for sharing status between the server and the client. For example, the client callback can be used for showing interim status of computation at the client machine and in interactive processing.
- **Cancellation of a session**
  Ninf-G2 provides a server-side API function named *ngstb_is_canceled()* for checking the arrival of cancel requests from the client. If the client calls a *grpc_cancel()* function, *ngstb_is_canceled()* returns *GRPC_TRUE*. In order to implement cancellation of a session, remote executables are required to call *ngstb_is_canceled()* at an appropriate interval and exit by itself if *ngstb_is_canceled()* returns *GRPC_TRUE*.

3.3.4. Solutions for Supporting Debugging of Applications Ninf-G2 provides functionalities which are useful for debugging. Ninf-G2 enables redirection of stdout and stderr of remote executables to the client machine. Log messages generated by Ninf-G2 and the Globus Toolkit can also be stored on the client machine. Furthermore, Ninf-G2 enables the launch of "gdb" on the server machine when a remote executable is launched on the server. These functionalities are made available by turning on the flags in the client configuration file.

4. Preliminary Performance Evaluation

In this paper, we evaluate the performance of Ninf-G2 focusing on the new feature that enables to create multiple function handles via a single GRAM call since this feature is essential for utilizing large-scale cluster of clusters. We have measured the invocation cost of remote executables, and performance using a weather forecasting system which is a typical parameter study application, using four clusters located at three geographically distributed sites.

4.1. Experimental Environment

We carried out experiments between AIST[1], TITECH[2], KISTI[3], and KU[4]. AIST is located in Tsukuba, which is approximately 50 miles northeast of Tokyo. Table 1 summarizes the configuration of the experimental environment. Six clusters (KOUME, UME, PrestoIII, Venus, Jupiter, and AMATA) are used in this experiment. A client program runs on the KOUME cluster and other clusters are used as servers. Latency and throughput in a TCP/IP communication are measured between the client machine (KOUME) and each server (see the "Latency" and "Throughput" rows in Table 1). Execution time of the weather prediction for a duration of one month on a single node at each server is also shown in the table (see the "1 month simulation" row). These values will be explained later.

4.2. Initialization of Function Handles

We measured the time for the initialization of function handles on the PrestoIII cluster. We varied the number of function handles from 1 to 200 and measured the time required for the initialization. We made the following two experiments. In the first experiment, we used the *grpc_function_handle_array_default_np()* function for initializing function handles, i.e. we utilized the new functionality of Ninf-G2. In the second experiment, we used the standard GridRPC API function (*grpc_function_handle_default()*). In order to initialize multiple function handles, *grpc_function_handle_default()* must be called for specific times in sequential. In the third experiment, we used multi-threaded version of the program used in the second experiment, i.e. all *grpc_function_handle_default()* functions are called in parallel, using multi-threading. These experiments indicate the effectiveness of this new feature of Ninf-G2 and the feasibility of using Ninf-G2 on large-scale clusters. The experimental results are shown in Figure 2.

If the functionality for initializing multiple function handles is used, the time for initialization of function handles depends slightly on the number of function handles, but the dependency is negligible. On the other hand, when we used the standard GridRPC API function, the time for initialization depends heavily on the number of function handles, even when we have multi-threaded the initialization. The time required for initializing ten function handles took three times longer than the time for one function handle. The time

1 National Institute of Advanced Industrial Science and Technology, Japan

2 Tokyo Institute of Technology, Japan

3 Korea Institute of Science and Technology Information, Korea

4 Kasetsart University, Thailand

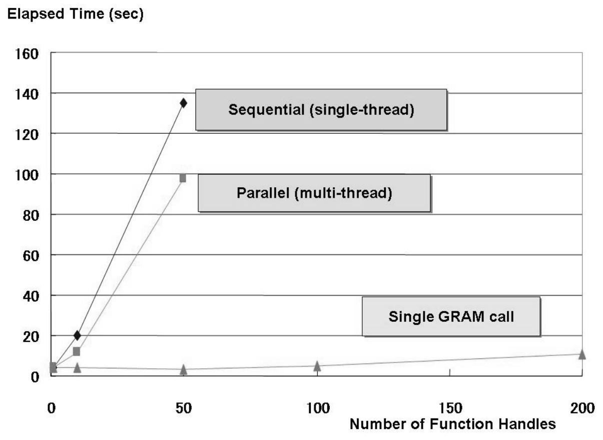

Figure 2. Time required for initializing function handles.

for fifty function handles took thirty times longer than the time for one function handle.

There are several reasons for this lowering of performance. One of the most important reason relates to the Globus jobmanager. Since a Globus jobmanager is invoked for initializing a function handle, as the number of function handles increases, the number of Globus jobmanagers increases as well, and invocation of many Globus jobmanagers increases the load of the front-end node, which lowers the performance of the front-end node. Indeed, when we initialized fifty function handles, the value of "load average" measured by the "uptime" command was more than 40. The experimental results show that the new functionality of Ninf-G2 enables the initialization of hundreds of function handles in a reasonable time, which is practically impossible using the standard GridRPC API function. Since the GridRPC API specifies primitive functions for GridRPC, it is desirable to provide high-level API functions, such as *grpc_function_handle_array_default_np()*, to provide higher performance and better functionality.

4.3. Weather Forecasting System

We measured the performance of Ninf-G2 using a weather forecasting system designed to predict short- to middle-term global weather change. The system was constructed by gridifying a weather prediction program called the Barotropic S-model [11] using Ninf-G2. The program executes hundreds to thousands of sample simulations while introducing perturbation for each simulation and tries to extend the predictability by taking a statistical ensemble mean of the simulation results. The simulation routine was installed as a "remote executable." We

Cluster	KOUME	UME	PrestoIII	Venus	Jupiter	AMATA
Site	AIST	AIST	TITECH	KISTI	KISTI	KU
Processor	Dual Pentium III 1.4GHz	Dual Pentium III 1.4GHz	Dual Athlon 1.6GHz	Pentium IV 2.0GHz	Pentium IV 1.7Ghz	Athlon 1.1GHz
Number of nodes	10	66	256	64	16	14
Jobmanager	–	SGE	PBS	PBS	PBS	SQMS
Latency (msec)	–	0.04	1.5	17.2	17.2	80
Throughput (MB/s)	–	99.0	8.2	0.98	0.98	0.09
1 month simulation (sec)	–	36	35	42	49	50

Table 1. Experimental Environment

have multi-threaded the client program. The client generates multiple threads, each of which corresponds to different servers, i.e. initialization of function handles and remote library calls on each server are processed by different threads. We used *grpc_function_handle_array_init_np()* for initializing multiple function handles on each server. We have deployed the remote executable for the simulation part in the UME, PrestoIII, and Venus clusters.

We measured the performance of the weather prediction for a duration of one month. We made the following three experiments.

1. The PrestoIII cluster was used as a server. We varied the number of processors: 1, 50, 150, and 200. The total number of sample simulations is specified as five times as many as the number of processors.
2. The PrestoIII and UME clusters were used as servers. We used 200 processors on the PrestoIII cluster and 50 processors on the UME cluster. 1250 sample simulations were executed.
3. The PrestoIII, UME, and Venus clusters were used as servers. We used 200 processors on the PrestoIII cluster, 50 processors on the UME cluster, and 50 processors on the Venus cluster. 1500 sample simulations were executed.

In all experiments, the client program ran on the KOUME cluster and we measured 1) the time required for transferring output data, 2) the execution time of each sample simulation, and 3) the total elapsed time of the weather prediction. The execution time of each sample simulation is the elapsed time between when the client starts data transfer to the remote executable and when the client receives results from the remote executable. The size of the input arguments (the data transferred from the client to a server) is approximately 3.5KB and the size of the output arguments (the data transferred from a server to the client) is approximately 400KB.

The experimental results are shown in Table 2. In the first experiment, as the number of processors increased, the time required for transferring output data also increased if the number of processors was less than one hundred, however the transfer time does not depend on the number of processors if the number of processors involved is more than one hundred. One possible reason for this behavior is as follows. As the number of processors increases, collisions of transfers of output data occur on the client machine. The collisions increase the data transfer time, but the collisions can be considered to become stable state phenomena if the number of processors involved is more than one hundred. Both the execution time of each sample simulation and the total elapsed time of the weather prediction increased as the number of processors increased. The overhead comes from the overhead of the initialization of function handles, but it is very small and is not proportional to the number of processors. When using 200 CPUs, the overhead for one sample simulation is less than 10%, and the total elapsed time increased by only 30 seconds. The efficiency is more than 80% in every case.

The second experiment generated the same result as the first experiment, but the result of the third experiment was different from those of the previous two experiments. All values increased drastically. Two possible reasons can be supposed, 1) data transfers between the client and servers exceeds the capacity of the network, i.e. the network itself becomes a bottleneck, and 2) computation on the client exceeds the capacity of the client machine, i.e. the client machine becomes a bottleneck. In order to investigate the reason, we carried out an additional experiment. In the additional experiment, we ran two client programs on the KOUME cluster. Each client program ran 750 sample simulations on the PrestoIII cluster (100 processors), the UME cluster (25 processors), and the Venus cluster (25 processors). The results are shown in the bottom in Table 3.

The performance was drastically improved when compared to the result of the third experiment. The result shows that the bottleneck was due to capacity of the client machine. This indicates that it is possible to run larger scale simulations by parallelizing the client program. There are

Exp. No.	No. of CPUs			No. of sample simulations	Data transfer time (sec)	Average exec. time of each simulation (sec)	Total elapsed time (sec)
	PrestoIII	UME	Venus				
Exp. #1	1	0	0	5	0.07	35.1	179
	50	0	0	250	0.36	36.1	192
	100	0	0	500	0.60	36.4	194
	150	0	0	750	0.74	36.9	199
	200	0	0	1000	0.60	38.1	205
Exp. #2	200	50	0	1250	0.53	37.6	225
Exp. #3	200	50	50	1500	11.40	53.8	450

Table 2. Results of Experiments 1, 2, and 3

No. of CPUs			No. of sample simulations	Data transfer time (sec)	Average exec. time of each simulation (sec)	Total elapsed time (sec)
PrestoIII	UME	Venus				
100+100	25+25	25+25	750+750	0.73	38.7	270

Table 3. Results of the additional Experiment

many possibilities for the parallelization, such as the use of MPI, OpenMP, multi-threading, and hierarchical RPC.

4.4. Comparison with Ninf-G version 1

We compared the performance of Ninf-G version 1 (Ninf-G1) and Ninf-G2 using the climate simulation. Since Ninf-G1 does not provide APIs for the creation of multiple function handles via a single GRAM call, this comparison give us effectiveness of the new feature of Ninf-G2. The UME, the Venus, the Jupiter, and the AMATA clusters were used as servers in the experiment. We ran 50 sample simulations for a duration of a hundred days on 10 processors (3 processors on the UME, the Venus, and the Jupiter, and 1 processor on the AMATA). The experimental results are shown in Figure 3. The total execution time was reduced from 1200 seconds to 850 seconds. We can figure out several reasons for this performance improvements. First, initialization cost was reduced by the new feature of creating multiple function handles via a single GRAM call. Second, data transfers were processed in parallel by multi-threading in Ninf-G2. Third, Ninf-G2 transfers data in binary format which provides better performance than XML format used in Ninf-G1. The experimental results indicate that the Ninf-G2 drastically improves the performance of Ninf-G1.

5. Demonstration at the SC2003

At the SC2003 Exhibition, we gave a demonstration of the weather forecasting system on the ApGrid [12], PRAGMA [13], and TeraGrid [14] Testbeds. We ran 1000

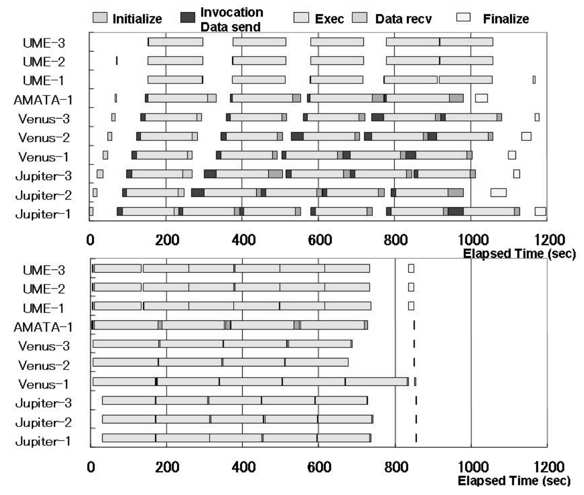

Figure 3. Execution profile of the weather simulation. The upper diagram shows the result of using Ninf-G1 and the lower shows that of using Ninf-G2.

sample simulations for a duration of ten days on the NCSA TeraGrid cluster (225 processors), the PrestoIII cluster (200 processors), the UME cluster (50 processors), and the Venus cluster (25 processors), i.e. 500 processors in total. The size of the input arguments (the data transferred from the client to a server) was approximately 3.5KB and the size of the output arguments (the data transferred from a server to the client) was approximately 140KB. The weather forecasting completed in approximately 150 seconds. One sample simulation for the same duration took approximately 12 seconds on a single node, i.e. it would take approximately 12,000 seconds (3 hours and 20 minutes) if we ran 1000 sample simulations on a single PC. The result is not sufficient from the point of view of “parallel efficiency,” however we were able to confirm that Ninf-G2 could stably utilize geographically distributed large-scale cluster of clusters with reasonable performance, even if the application has relatively fine-grain parallelism. The results indicate the feasibility of Ninf-G2 for development and execution of Grid applications on a large-scale computational Grid.

6. Conclusions and Future Work

This paper reports on the design, implementation, and preliminary performance evaluation of a GridRPC system called Ninf-G2. Ninf-G2 is designed so that it provides 1) high performance on a large-scale computational Grid, 2) rich functionalities which are required to compensate for the heterogeneity and unreliability of a Grid environment, and 3) an API which supports easy development and execution of Grid applications.

The experimental results showed that the new functionalities and implementation of Ninf-G2 enables stable and efficient utilization of large-scale cluster of clusters. Through the demonstration at the SC2003 exhibition, we were able to confirm the feasibility of Ninf-G2 on large-scale cluster of clusters, and to confirm that Ninf-G2 can give opportunities to run on a computational Grid with realistic performance for relatively fine-grain task-parallel applications which are considered “non-attractive” applications on a Grid.

As of the beginning of September 2004, Ninf-G version 2.2.0 is available on the home page of the Ninf Project [3]. At the next stage, we will evaluate the performance of Ninf-G2 on larger-scale computational Grids, which are expected to include the AIST Super Cluster (2048 processors), ApGrid, PRAGMA and the TeraGrid. We will also evaluate effectiveness and usability of the new features of Ninf-G2 such as “Ninf-G remote objects” and “Client Callbacks.”

Acknowledgment

We would like to give thanks to Dr. Tanaka in University of Tsukuba. He was kind enough to provide his S-model program for our research work.

This work has been conducted as part of ApGrid and PRAGMA activities. We wish to express our gratitude to all of the ApGrid and PRAGMA participants, especially TITECH, KISTI, and KU for their resource contribution for our experiments, and also APAN members for setting up network configurations in the Asia-Pacific region. We were able to give a demonstration of our system at the SC2003 with the help of Tera Grid Executive Committee members and Tera Grid help team.

A part of this research was supported by a grant from the Ministry of Education, Sports, Culture, Science and Technology (MEXT) of Japan through the NAREGI (National Research Grid Initiative) Project.

References

[1] Henri Casanova and Jack Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. In *Proceedings of Super Computing ’96*, 1996.

[2] Hidemoto Nakada, Mitsuhisa Sato, and Satoshi Sekiguchi. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15:649–658, 1999.

[3] Ninf. http://ninf.apgrid.org/.

[4] Yoshio Tanaka, Hidemoto Nakada, Satoshi Sekiguchi, Toyotaro Suzumura, and Satoshi Matsuoka. Ninf-g: A reference implementation of rpc-based programming middleware for grid computing. *Journal of Grid Computing*, 1(1):41–51, 2003.

[5] Eddy Caron, Frederic Desprez, Frederic Lombard, Jean-Marc Nicod, Martin Quinson, and Frederic Suter. A scalable approach to network enabled servers. In *Proceedings of the 8th International EuroPar Conference*, volume 2400 of Lecture Notes in Computer Science, pages 907–910. Springer Verlag, 2002.

[6] Mitsuhisa Sato, Taisuke Boku, and Daisuke Takahashi. Omnirpc: A gridrpc system for parallel programming in cluster and grid environment. In *3rd International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pages 206–213, 2003.

[7] Henri Casanova, Thomas Bartol, Joel Stiles, and Francine Berman. Distributing mcell simulations on the grid. *Journal of Supercomputing Applications*, 15(3):243–257, 2001.

[8] Hiroshi Takemiya, Yoshio Tanaka, Kazuyuki Shudo, and Satoshi Sekiguchi. Constructing grid applications using standard grid middleware. *Journal of Grid Computing*, 1(2):117–131, 2003.

[9] Kento Aida, Wataru Natsume, and Yoshiaki Futakata. Distributed computing with hierarchical master-worker paradigm for parallel branch and bound algorithm. In *3rd International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pages 156–163, 2003.

[10] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 1997.

[11] Hiroshi. L. Tanaka and D. Nohara. A study of deterministic predictability for the barotropic component of the atmosphere. Science Reports of the Institute of Geoscience, University of Tsukuba, 2001.

[12] Asia Pacific Partnership for Grid Computing. http://www.apgrid.org/.

[13] Pacific Rim Applications and Grid Middleware Assembly. http://www.pragma-grid.net/.

[14] TeraGrid. http://www.teragrid.org/.