Optimization Problem Solving System using GridRPC

Hisashi Shimosaka, Tomoyuki Hiroyasu, *Member, IEEE,*
Mitsunori Miki, *Member, IEEE,* and Jack Dongarra, *Member, IEEE,*

Abstract

In recent years, the Grid has been standardized with an emphasis on efficient services integration over the wide area network. We focus on applications integration for solving complex optimization problems on the Grid. Then, the framework of applications integration using the GridRPC is proposed. In this framework, each service on the Grid has four basic functions that enable an end-user to invoke applications and to exchange information among applications. By application holders expressing their own applications as a service along the framework, end-users are able to design arbitrary applications integration through the wide area network. In addition, we also propose the Application Programming Inferfaces (APIs) that enable the service to invoke the designed applications integration. These APIs support the creation of an optimization problem solving system. Therefore, end-users can solve their own optimization problems by designing applications integration on the system.

We implement the proposed framework and APIs using the NetSolve system. Through computation simulation experiments of our implementation of the system, we found that the overhead time is very short compared to the calculation time of optimization and that the system can be used practically in the wide area network. Therefore, the proposed framework and APIs can be expected to be efficient in the system construction for optimization problem solving on the Grid.

Index Terms

Grid, GridRPC, NetSolve, Applications Integration, Optimization Problem Solving.

H. Shimosaka is a Ph.D. Candidate at the Graduate School of Engineering, Doshisha University, Japan. E-mail: hisashi@mikilab.doshisha.ac.jp

T. Hiroyasu is with the Department of Knowledge Engineering, Doshisha University, Japan. E-mail: tomo@is.doshisha.ac.jp

M. Miki is with the Department of Knowledge Engineering, Doshisha University, Japan. E-mail: mmiki@mail.doshisha.ac.jp

J. Dongarra is with the Computer Science Department, University of Tennessee, USA. E-mail: dongarra@cs.utk.edu

Optimization Problem Solving System using GridRPC

I. Introduction

In the field of multidisciplinary design optimization and analysis such as automobile or aerospace design, several types of sophisticated applications are needed for structural analysis, fluid dynamics, optimization, visualization and so on. In addition, these applications should be efficiently integrated while maintaining consistency among applications. In a general system used to solve complicated optimization problems, application holders register existing applications as a service on the system in advance. Then, end-users realize their own problem solving by aggregating applications information and designing arbitrary applications integration. On the other hand, because the private device, database and/or parallel computer may be required in application execution and computing performance can increase by performance tuning suited to the computing environment, the approach of building an optimization problem solving system through the wide area network is often introduced[1], [2].

In the system construction through the wide area network, it is indispensable to use the Grid[3], [4], [5], which can seamlessly and safely integrate disparate calculation and information resources from the perspective of development cost and system practicality. Many Grid middleware, which support a Grid-enabled application are already developed[6], [7], [8], [9], [10] and some of them are beginning to be used in science fields. In recent years, the Service Oriented Architecture (SOA) has been adopted as the basic concept of the Grid technologies and the Open Grid Services Architecture (OGSA)[5], [11] has been proposed at the Global Grid Forum (GGF)[12]. The OGSA has subsequently been standardized with the emphasis on efficient integration of several services on the wide area network.

Because of these backgrounds, we especially focus on applications integration and optimization problem solving on the Grid. Then the framework of applications integration using the GridRPC[10] is proposed. The GridRPC is one of the useful programming models on the Grid

and it has some sophisticated Grid middleware that implement the GridRPC APIs[13], [14]. As application holders express existing applications as services along the proposed framework, end-users are able to design arbitrary applications integration through the wide area network. In addition, we also propose the APIs that enable services to invoke the applications integration designed by end-users. The proposed APIs support to construct an optimization problem solving system on the Grid. Therefore, end-users are also able to solve their own optimization problems by designing applications integration.

In the computational simulation examples detailed in this paper, we constructed the optimization problem solving system using the NetSolve system[13], which is one of the GridRPC systems. The performance of the system was evaluated and is discussed here.

II. Grid and GridRPC

A. Overview of the Grid

In recent years through improvements in wide area network performance, the Grid has produced high expectations for its potential as a next generation cyberinfrastructure. It is expected that several systems, which were not able to be realized previously can now be built easily[3], [4]. Most research on the Grid involve the development of several types of Grid middleware that solve various problems caused by sharing resources through the wide area network such as user authorization/authentication, communication encryption, job scheduling, fault tolerance, information service and programming model. Typical examples of such middleware include the Globus Toolkit[6], currently developed by the Globus Alliance[15]; Legion[7], which provides basic services and tools; Condor[9] and AppLes[8], which provide dynamic scheduling function; and the GridRPC systems whose details are explained later. Most of the Grid technologies are standardized at the GGF. Recently, in order to promote Grid technologies for business, the SOA has been adopted as the basic concept of the Grid technologies and the OGSA was proposed and standardized at the GGF. In the OGSA, these have been standardized with an emphasis on safe and efficient integration among services arranged on the wide area network using the Grid technologies. Since this tendency will be increasingly accelerated, it is also important to develop the optimization problem solving system on the Grid with the emphasis on services integration.

B. GridRPC

The GridRPC, which extends a Remote Procedure Call (RPC) mechanism for the Grid, is a typical programming model for develop-

ing Grid-enabled applications. In the GridRPC, application holders arrange their own applications on remote servers in advance. Then, end-users invoke one of them through the wide area network by using the GridRPC APIs, which have been standardized at the GGF. The GridRPC has two advantages. First, it has excellent usability for end-users. Second, it enables application holders to arrange and publish existing applications on the Grid easily and safely. In addition, some sophisticated systems that implement the GridRPC APIs such as NetSolve, Ninf[14] and Ninf-G are already developed. Therefore, it is expected that the GridRPC system will be widely used.

C. NetSolve

The NetSolve system is mentioned as one of the sophisticated GridRPC systems and we illustrate the overview of the NetSolve system in Fig.1. NetSolve consists of three major components; the NetSolve server, the NetSolve agent and the NetSolve client. The NetSolve server manages and performs applications prepared by application holders. The NetSolve agent unifies a dataset on the information of the NetSolve server and mediates a client request. The NetSolve client requests one of the published applications on the NetSolve servers through the NetSolve agent. Of these components, the most important is the agent. In the NetSolve system, each server registers its capabilities such as prepared applications and hardware performance with the agent in advance. Then, the NetSolve agent periodically checks the condition of the NetSolve servers. As a result, the NetSolve client can obtain information about all the available applications from the NetSolve agent. In addition, this mechanism enables the NetSolve client to invoke one of the published applications on the most appropriate NetSolve server. When the client requests the application through the agent, the agent selects the most appropriate server based on the server's capabilities and returns the information of the selected server. Then the NetSolve client invokes the application with the input data to the selected server. After execution of the application, the server returns the result data to the client. After the request to the NetSolve agent is automatically performed, the set of jobs is performed without the intervention of the end-user. These features of the NetSolve system are suitable for a system construction based on the SOA. This means that end-users are able to easily and efficiently use existing applications published by application holders as a service through the wide area network.

III. Optimization problem solving system

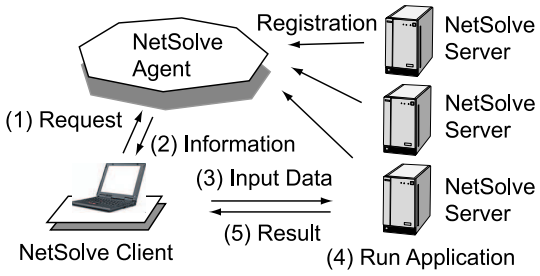

Fig. 1. Overview of the NetSolve system

USING THE GRIDRPC

In this paper, we propose the framework of applications integration using the GridRPC. In addition, in order to support to construct an optimization problem solving system on the Grid, the APIs for invoking applications integration are also proposed. By application holders expressing existing applications as the service using the proposed framework and APIs, end-users are able to solve their own optimization problems by designing arbitrary applications integration through the wide area network. In this section, we firstly describe the optimization problem solving system using the GridRPC, which extends the features of the NetSolve system. And then, the proposed framework and APIs are explained with examples of applications integration for solving optimization problems.

A. Requirement

Application executions are generally regarded as the generation of output files from input files. In order to minimize the modification of the source code of the application for applications integration, it is more practical to integrate applications by exchanging input/output files among applications.

In the general system for solvint optimization problems, the optimization system is assumed as shown in Fig.2(a). This system consists of two parts, optimizing service and analyzing service. The optimizing service determines a next searching point and optimizes design variables. The analyzing service analyzes values of an objective function and constraints. Whenever the analyzed values that correspond to a search point are required in the optimizing service, the optimizing service obtains the values by invoking the analyzing service. The simplest optimization system is a system that assigns one application to each service. However, depending on the kind of optimization problem, it is often necessary to assign two or more applications integration and/or another optimization system to the analyzing service as shown in Fig.2(b). Therefore, we consider that the optimization problem solving system should have the following functions for end-users to solve their own optimization problems by arbitrarily integrating applications.

1) A function where an end-user can design arbitrary applications integration as the analyzing service.

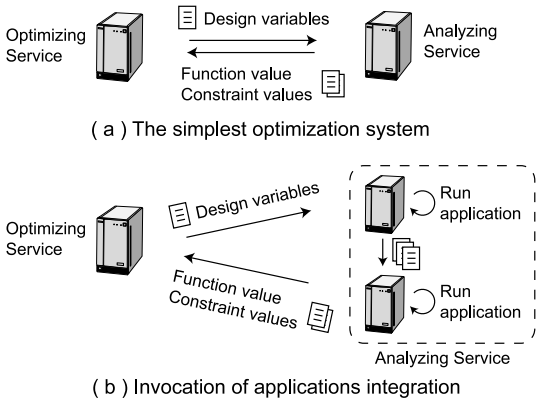

Fig. 2. Optimizing service and analyzing service

2) A function where an end-user can specify the designed applications integration to the optimizing service and the optimizing service can invoke it as the analyzing service.

In applications integration within science fields, the exchange of input/output files among services has a tendency to create a bottleneck because of large sizes. Therefore, the above functions should be realized by decentralized administration among services.

B. Overview of the optimization problem solving system

The overview of the optimization problem solving system using the GridRPC is shown in Fig.3. In order to extend the features of the NetSolve system, this system consists of three components, client, services and agent similar to the NetSolve system. The client designs applications integration and builds an optimization system to solve an optimization problem. Each service manages and performs applications prepared by application holders. In addition, each service also provides the four basic functions to realize applications integration by decentralized administration among services. The agent unifies a dataset of the services information and mediates a client request.

In this system, the client builds the optimization system in accordance with the following steps. First, the client aggregates the information of all the available applications from the agent. Second, the client selects the services and designs applications integration by deciding the invocation sequence of the functions provided by each selected service. Finally, the client requests to perform the functions in a suitable order through the agent.

In addition, applications integration on the system is realized by performing applications on the services and exchanging information among services. The information exchange is also realized by sending and receiving input/output files of applications. Therefore, the four basic functions of each service are prepared for the client to design applications integration. These functions are implemented by the GridRPC. Thus, the client calls some of the GridRPC requests to solve optimization problems.

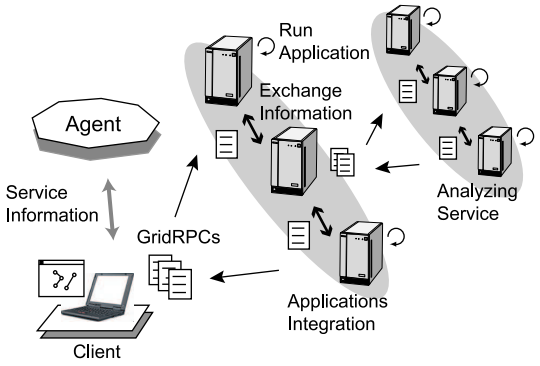

Fig. 3. Overview of the optimization problem solving system

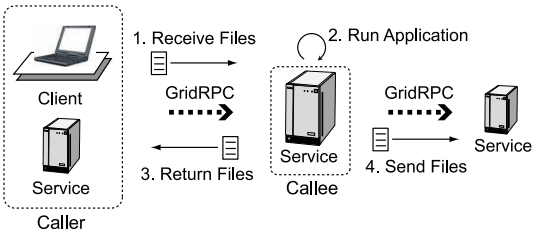

Fig. 4. Four basic functions provided by each service

C. Basic functions of each service

In order to realize applications integration, each service provides the four basic functions to the client. The outline of these functions is shown in Fig.4. These functions are implemented by the GridRPC and the details of these functions are shown below.

1) "Receive Files" is the function to receive input files of an application from a caller of the GridRPC.
2) "Run Application" is the function to perform an application specified by a caller of the GridRPC.
3) "Return Files" is the function to send output files generated by performing an application to a caller of the GridRPC.
4) "Send Files" is the function to send output files generated by performing an application to another service specified by a caller of the GridRPC.

In these functions, the Receive Files, Run Application and Return Files are realized by one GridRPC request and only Send files is realized by two requests.

D. Desing of the applications integration

We show an example of applications integration using the basic functions provided by each service in Fig.5, which also shows that a client designs applications integration between two services. In this example, the client first aggregates the information of all the available applications from an agent and selects the Service A and B. Then, the client prepares only the input file of Service A because the input file of Service B is offered by the output file of Service A. Finally, the client also prepares the configuration file describing the following jobs and integrates the applications by making the GridRPC requests. These jobs are invoked through a total of six GridRPC requests, which consists of three requests from the client to Service A, one request from Service A to

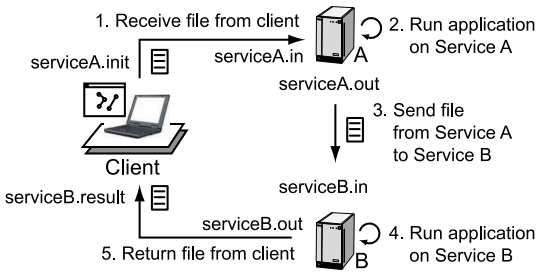

Fig. 5. Applications integration example

```
NUMBER OF JOBS = 5;
JOBS TYPE = { 1,2,4,2,3 };
CALLERS = { localhost,localhost,serviceA,localhost,localhost };
CALLEES = { serviceA,serviceA,serviceB,serviceB,serviceB };
NUMBERS OF FILES = {1,0,1,0,1};
CALLER'S FILES = { serviceA.init };
CALLEE'S FILES = { serviceA.in };
CALLER'S FILES = { };
CALLEE'S FILES = { };
CALLER'S FILES = { serviceA.out };
CALLEE'S FILES = { serviceB.in };
CALLER'S FILES = { };
CALLEE'S FILES = { };
CALLER'S FILES = { serviceB.result };
CALLEE'S FILES = { serviceB.out };
```

Fig. 6. Configuration file example

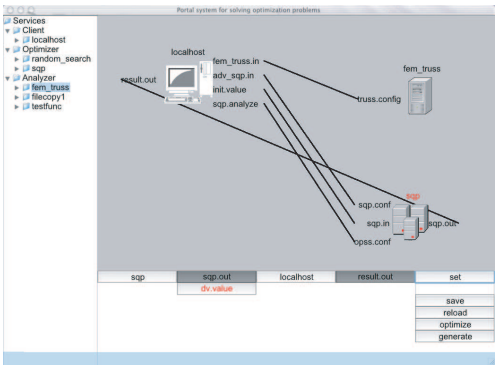

Fig. 7. User interface tool

Service B and two requests from the client to Service B.

1) Service A receives the input file from the client (Service A: Receive Files).
2) Service A performs the managing application (Service A: Run Application).
3) Service A sends the output file to Service B (Service A: Send Files).
4) Service B performs the managing application (Service B: Run Application).
5) Service B sends the output file to the client (Service B: Return Files).

We also show the example of the configuration file in Fig.6. This configuration file shows the above applications integration in our implementation of the system. In the configuration file, an end-user describes the number of the job requests and the type, the caller/callee, the number of sent/received files and their file names of each request. Moreover, in order to reduce the burden of the client, our implementation provides the user interface tool shown in Fig.7. This tool enables an end-user to aggregate the information of applications from the agent, to easily express the relations of input/output files among services, and to automatically generate the configuration file.

E. Application Programming Interface

As described in Sec.III-A, in order to solve optimization problems by integrating applications, it is indispensable for an end-user to specify the designed applications integration to the optimizing service. In addition, the optimizing service has to invoke it as the analyzing ser-

vice. Therefore, we propose some of the APIs that enable application developers to easily use it in their own applications. By substituting parts where require analyzed values that correspond to a searching point with these APIs, the optimizing service can read the configuration file generated by an end-user and invoke it as the analyzing service.

1) Include file: The name of the include file is defined as "opss.h". All programs which use one of the APIs have to include this file.

2) Status code: All of the APIs return the status code as the return value of a function. Whenever a function is completed normally, " OPSS_OK " is returned and whenever an error occurs, " OPSS_FAILURE " is returned.

3) Initializing and finalizing functions: The initializing API initializes variables, the required modules and the GridRPC middleware. The finalizing API releases any resources and finalizes the Grid middleware. The "OPSS" used in these APIs is one of the structures for maintaining the status of the system.

```
int grpc_opss_initialize( OPSS *pt, char *agent_name );
int grpc_opss_finalize( OPSS *pt );
```

4) Configuration file specification function: The following API can specify the configuration file used for applications integration. The purpose of this API is to invoke the specific applications integration predefined by the application developer's own request.

```
int grpc_opss_config( OPSS *pt, char *configuration_filename );
```

5) Application integration functions: In order to invoke applications integration, any of the following two APIs have to be used. The former API assumes being mainly used in analyzing applications. Whenever the former API is used, it is necessary to specify the configuration file using the latter API. The latter API assumes being mainly used in optimizing applications. The application developers of optimizing applications have to substitute parts that require analyzed values with this API. Whenever the API is used, it is not necessary for an application developer to specify the configuration file. The configuration file provided by an end-user is automatically used instead.

```
int grpc_opss_optimize( OPSS *pt );
int grpc_opss_analyze( OPSS *pt );
```

F. Construction of the optimization system

In this section, we show an example of the structural optimization system. An overview of this example is illustrated in Fig.8. In this example, the truss structural application using the finite element method (fem_truss) is used as the analyzing service and the sequential

quadratic programming method (sqp) is used as the optimizing service.

1) Input and output files: The fem_truss requires truss.config and truss.in as the input files and generates truss.out as the output file. In the truss.config, the default parameters of analysis and the basic structure of truss are defined. In the truss.in, the cross-section areas of each member are described. These areas correspond to design variables sent from the sqp to the fem_truss. In the truss.out, analyzed values such as the weight value of the truss structure and the stress values of each member are described. The weight value corresponds to an objective function and the stress values correspond to constraints. These values are sent from the fem_truss to the sqp.

On the other hand, the sqp requires sqp.config and sqp.in as the input file and generates sqp.out as the output file. In the sqp.config, the default parameters of optimization are described and the initial searching point is described in the sqp.in. In the sqp.out, the optimization result is described. In addition, whenever analyzed values that correspond to design variables are required in the sqp, the sqp writes the values of design variables into the sqp.dv and invokes the applications integration as the analyzing service. After invoking it, the sqp reads the values of the objective function and constraints from the sqp.obj. In order

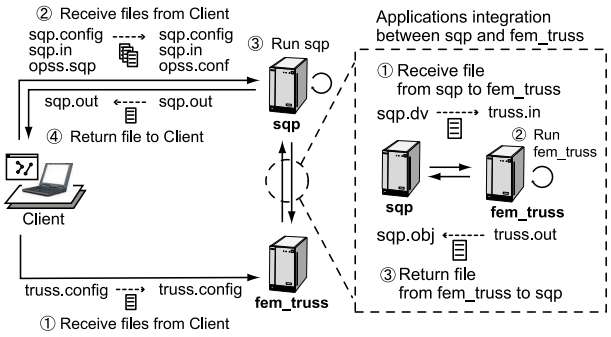

Fig. 8. Optimization system example

to enable applications integration between the fem_truss and the sqp, it is assumed that the sqp.dv is described by the same format as the truss.in and the sqp.obj is also described by the same format as the truss.out.

2) Design of the optimization system: In order to construct the optimization system shown in Fig.8, the client should prepare the configuration file described by the following jobs in descending order. These jobs are invoked by a total of six GridRPC requests that consist of one request from the client to the fem_truss and five requests from the client to the sqp. In our implementation of the system, the client needs to call many requests to the sqp because the GridRPC request can not transfer multiple files.

1) The client sends the truss.config to the fem_truss. (fem_truss: Receive Files)
2) The client sends the sqp.config, sqp.in and opss.conf to the sqp. (sqp: Receive Files)

3) The sqp performs the sqp application. (sqp: Run Application)
4) The sqp returns the sqp.out to the client. (sqp: Return Files)

On the other hand, whenever the sqp requires analyzed values that correspond to design variables, the sqp should perform the following jobs in descending order. These jobs are invoked by three GridRPC requests from the sqp to the fem_truss and are repeatedly performed until the optimization is finished.

1) The sqp sends the sqp.dv to the fem_truss. (fem_truss: Receive Files)
2) The fem_truss performs the fem_truss application. (fem_truss: Run Application)
3) The fem_truss returns the truss.out to the sqp. (fem_truss: Return Files)

In order to perform these jobs on the sqp, the client has to describe another configuration file and send it to the sqp before performing the sqp application (sqp: Run Application). The sqp application reads the configuration file and invokes the applications integration as the analyzing service. Therefore, as in Fig.8, the client sends the " opss.conf " to the sqp before performing the sqp application. On the other hand, in the sqp application, in order to invoke the applications integration predefined by the client, the application developer has to substitute parts that require analyzed values with the "grpc_opss_analyze" function described in Sec.III-E.5 in advance.

IV. Computational simulation experiments

A. Performance evaluation of the basic functions

1) Overview of the experiment: In this experiment, we evaluated the basic performance of our implementation of the system using the applications integration shown in Fig.5. In order to correctly evaluate the basic performance, both Service A and B manage the same application which replicates the input file into the output file. In addition, no content is described in the input file of Service A. Thus, the exchanged file size is minimized.

In the experiment, the client first described the configuration file shown in Fig.6 and saved it as the file name " opss.conf ". The client also described the client program shown in Fig.9 using the APIs described in Sec.III-E. The execution time of each API used in Fig.9 was measured in this experiment. In addition, in the "grpc_opss_optimize" function which invokes the applications integration, the execution time of each basic function described in Fig.6 was also measured.

A cluster environment connected via fast Ethernet was used for the experiment and each single node was assigned to the client, the

```
#include "opss.h"
int main(){

  OPSS op;

  if( grpc_opss_initialize( &op, "agent" ) != OPSS_OK){  exit(2);  }
  if( grpc_opss_config( &op, "opss.conf" ) != OPSS_OK){  exit(2);  }
  if( grpc_opss_optimize( &op ) != OPSS_OK){  exit(2);  }
  if( grpc_opss_finalize( &op ) != OPSS_OK){  exit(2);  }

  return 0;
}
```

Fig. 9. Client program used for the numetical example

TABLE I

ELAPSED TIME OF EACH API ON THE CLUSTER ENVIRONMENT

API	Elapsed Time (s)
grpc_opss_initialize	1.009
grpc_opss_config	0.002
grpc_opss_optimize	6.169
grpc_opss_finalize	4.450

agent, and the Services A and B. Moreover, the NetSolve system (ver.2.0) was used for the GridRPC middleware of our implementation.

2) Result of the experiment: The average execution time of each API in 100 trials is shown in Table.I. We found that the initialization of the system required 1.0 seconds, the finalization required 4.5 seconds and the applications integration, which consisted of the five GridRPC requests from the client, required 6.2 seconds. The average execution time of each basic function provided by the services is also shown in Table.II. All of the functions except the Send Files could be performed in less than 0.06 seconds. However, the execution of the Send Files required 6.0 seconds. This was caused by the implementation of the Send Files. The Send Files are mainly implemented by three parts; the initialization of the system, the Receive Files function invocation of the service specified by the client, and the finalization of the system.

TABLE II

ELAPSED TIME OF EACH SERVICE FUNCTION ON THE CLUSTER ENVIRONMENT

Function	Elapsed Time (s)
Service A: Receive Files	0.058
Service A: Run Application	0.017
Service A: Send files	6.023
Service B: Run Application	0.017
Service B: Return Files	0.054

From these results, we conclude that the "grpc_opss_finalize", which finalizes the system, requires the longest time in the APIs and the basic functions. This was caused by the implementation of the NetSolve system. On the other hand, since the execution time of the basic functions except the Send Files was less than 0.06 seconds, we found that the overhead time of the proposed system was very short.

B. Performance evaluation of the optimization system

1) Overview of the experiment: In this experiment, the performance of the optimization system as shown in Fig.8 was evaluated. As with the last experiment, the client firstly described the configuration file and saved it as the file name " opss.conf ". And then, the client performed the optimization by using the client program shown in Fig.9. The performance of the optimization system was evaluated by measuring the execution time of each API and each basic function of the server. As the experimental environments, the cluster environment connected via fast Ethernet and the Grid environment consisted of clusters at Doshisha University, Japan and the University of Tennessee, USA. The Grid environment is summarized in Fig.10. Both directions of the network performance between the sqp and the fem_truss was 94.1Mbps in the cluster environment and 2.2Mbps in the Grid environment when the TCP stream performance was measured by sending and receiving 1KB message using the Netperf benchmark[16].

2) Result of the experiment: The average execution time of each API in 30 trials is shown in Table.III. In addition, the average execution time of each basic function provided by the sqp and fem_truss services is also shown in

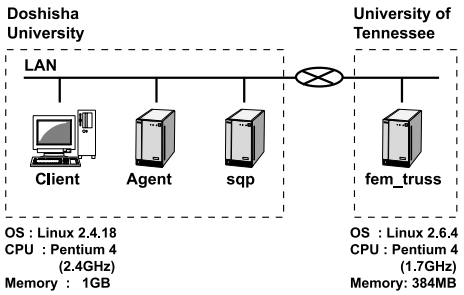

Fig. 10. Specification of the Grid environment for the numerical example

Table.IV. In the execution of the four APIs to perform the optimization, it was found that the Grid environment required more time than the cluster environment. It was also found that the execution time of the Run Application, which invokes the sqp application, made up more than 99% of the total execution time. This was caused by the difference in the communication performance of the NetSolve system between the Grid environment and the cluster environment when the sqp called the GridRPC requests to the fem_truss.

On the other hand, in the optimization system used for this experiment, the optimizing service (sqp service) invoked the analyzing service (fem_truss service) 246 times. Therefore, each invocation, which consisted of three GridRPC requests, required 0.14 seconds in the cluster environment and 8.14 seconds in the Grid environment on average. Since the scale of the target problem used for this experiment

TABLE III

ELAPSED TIME OF EACH API ON THE CLUSER AND THE GRID ENVIRONMENT

API	Cluster (s)	Grid (s)
grpc_opss_initialize	1.009	1.010
grpc_opss_config	0.002	0.002
grpc_opss_optimize	35.723	2005.795
grpc_opss_finalize	4.373	5.045

TABLE IV

ELAPSED TIME OF EACH SERVICE FUNCTION ON THE CLUSER AND THE GRID ENVIRONMENT

Function	Cluster (s)	Grid (s)
fem_truss: Receive Files	0.053	2.689
sqp: Receive Files	0.155	0.154
sqp: Run Application	35.461	2002.893
sqp: Return Files	0.054	0.059

was very small, all of them were regarded as the overhead time of the system. However, as the scale of the target problem becomes larger and the execution time of the analyzing service is longer, the rate of the overhead time of the system becomes shorter. If the average execution time of the analyzing service is 30.0 seconds, the rate of the overhead time of the system is about 0.5% in the cluster environment and 21.3% in the Grid environment. Therefore, in the wide area network where communication performance is poor, the system can be used practically.

V. RELATED WORKS

As related work of our optimization problem solving system using the GridRPC, NEOS (Network-Enabled Optimization System)[1], developed at the Argonne National Laboratory, and the FIPER system[2], developed under the FIPER Project, are mentioned. NEOS is one of the systems based on Condor. In NEOS, an end-user defines an optimization problem and performs the jobs by selecting an optimization method. In order to solve the defined problem, Condor allocates the jobs to idle resources on a network. On the other hand, the FIPER system consists of the ACS server (Application Control System Server) and some of the ACS stations, which are based on the Java 2 Platform, Enterprise Edition. In order to construct an optimization system in the FIPER system, an end-user integrates components of each FIPER station centering on the ASC server by arbitrarily describing a workflow and managing inputs/outputs of components. Either can build an optimization system by only selecting services on a network without an end-user having optimizing applications. However, in the NEOS, because the relation between optimizing application and

analyzing application is fixed, an end-user can not arbitrarily integrate applications. In addition, since the FIPER system is based on web service technologies and has a central control mechanism as the ACS server, the proposed system, which is based on the GridRPC and does not have a central control mechanism, is better in safety and scalability.

VI. Conclusion and future works

In recent years, Grid technologies have been standardized with the emphasis on efficient services integration on the wide area network. We especially focused on applications integration for optimization problem solving on the Grid. Then, the framework of applications integration using the GridRPC is proposed. By application holders expressing their own applications as the service along the framework, end-users are able to design arbitrary applications integration through the wide area network. In addition, we also proposed the APIs that enable the service to invoke the applications integration designed by the end-user. The proposed framework and APIs support the construction of the optimization problem solving system on the Grid. Therefore, end-users can solve their own optimization problems by designing applications integration on the system.

Through the computation simulation experiments of our implementation of the system, we conclude that the finalization of the system takes longer compared to other functions because of the implementation of the NetSolve system. In addition, we also found that the overhead time of the system in the cluster environment is very short. The system can be also used in the wide area network where communication performance is poor. In future work, we will implement the system using other GridRPC middleware such as the Ninf-G and compare the performance of the systems. In addition, the functions to invoke analyzing services in parallel and to translate the format of input/output files exchanged among applications are also necessary.

Acknowledgment

We wish to thank Mr. Keiji Kudo (Engineous Japan Inc.) who provided much advice about the FIPER system. We also thank Mr. Yutaka Ueshima (Japan Atomic Energy Research Institute) who provided many ideas for the application integration system on the wide area network. Finally, this research was partially supported by a grant from the Information-technology Promotion Agency (IPA), Japan under the Mito Software Creation Program.

References

[1] J. Czyzyk, M. P. Mesnier, and J. J. Moré, “NEOS: The Network-Enabled Optimization System,” Mathemat-

ics and Computer Science Division, Argonne National Laboratory, 1996.

[2] P. J. Róhl, R. M. Kolonay, R. K. Irani, M. Sobolewski, K. Kao, and M. W. Bailey, "A Federated Intelligent Product EnviRonment," *Proceedings of the 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2000.

[3] I. Foster and C. Kesseleman, *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[4] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid : Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, 2001.

[5] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Globus Project, 2002. [Online]. Available: http://www.globus.org/alliance/publications/papers/ogsa.pdf

[6] I. Foster and et.al., "Globus : A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, 1997.

[7] A. Grimshaw, W. Wulf, and the Legion team, "The Legion Vision of a Worldwide Virtual Computer," *Communications of the ACM*, vol. 40(1), pp. 39–45, 1997.

[8] H. Casanova, G. Obertelli, F. Berman, and R. Wolski, "The AppLeS Parameter Sweep Templete : the User-Level Middleware for the Grid," *Proceedings of the Supercomputing '00 Conference*, 2000.

[9] M. Litzkow, M. Livny, and W. Mutka, "Condor - A Hunter of Idle Workstations," *Proceedings of the 8th International Conference on Distributed Computing Systems*, pp. 101–111, 1988.

[10] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova, "Grid RPC : A Remote Procedure Call API for Grid Computing," Innovative Computing Laboratory, University of Tennessee, 2002.

[11] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich, "The Open Grid Services Architecture, Version 1.0," Global Grid Forum OGSA-WG, 2005. [Online]. Available: http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf

[12] "Global Grid Forum." [Online]. Available: http://www.gridforum.org/

[13] H. Casanova and J. Dongarra, "NetSolve: A Network Server for Solving Computational Science Problems," *Proceedings of the Supercomputing '96 Conference*, 1996.

[14] H. Nakada, M. Sato, and S. Sekiguchi, "Design and Implementations of Ninf : towards a Global Computing Infrastructure," *Future Generation Computing Systems, Meta-computing Issue*, 1999.

[15] "Globus Alliance." [Online]. Available: http://www.globus.org/

[16] "Netperf benchmark." [Online]. Available: http://www.netperf.org/

Hisashi Shimosaka Biography text here.

Tomoyuki Hitoyasu Biography text here.

PLACE
PHOTO
HERE

Mitsunori Miki Biography text here.

PLACE
PHOTO
HERE

Jack Dongarra Biography text here.