GrADSolve - RPC for High Performance Computing on the Grid *

Sathish S. Vadhiyar and Jack J. Dongarra

Computer Science Department
University of Tennessee, Knoxville
{vss, dongarra}@cs.utk.edu

Abstract. Although high performance computing has been achieved over computational Grids using various techniques, the support for high performance computing on the Grids using Remote Procedure Call (RPC) mechanisms is fairly limited. In this paper, we discuss a RPC system called GrADSolve that supports execution of parallel applications over Grid resources. In GrADSolve, the resources used for the execution of parallel application are chosen dynamically based on the load characteristics of the machines. Also, GrADSolve stages the user's data to the end resources based on the data distribution used by the end application. Finally, GrADSolve allows the users to store execution traces for problem solving and use the traces for subsequent solutions. Experiments are presented to prove that GrADSolve's data staging mechanisms can significantly reduce the overhead associated with data movement in current RPC systems. Results are also presented to demonstrate the usefulness of utilizing execution traces maintained by GrADSolve for problem solving.

1 Introduction

The role of Remote Procedure Call (RPC) [2, 16, 20, 4, 9, 3, 13] mechanisms in Computation Grids [7] has been the subject of recent studies [15, 6, 14]. Computational Grids consists of large number of machines ranging from workstations to supercomputers and strive to provide transparency to the end users and high performance for end applications. While high performance is achieved by the parallel execution of applications on large number of Grid resources, user transparency can be achieving by employing RPC mechanisms. Though there are large number of RPC systems that adequately support the remote invocation of sequential software from sequential environments, the number of RPC systems for supporting invocation of parallel software are relatively few [3, 13, 14, 10, 5]. Some of these parallel RPC systems [14, 10] require invocation of remote parallel services from only parallel clients. Some of the RPC systems [3, 13] support only master-slave or task farming models of parallelism. Few RPC systems [13, 3] fix the amount of parallelism at the time when the services are uploaded into the

RPC system and hence are not adaptive to the load dynamics of the Grid resources. Few RPC systems [10, 5] supporting invocation of parallel software are implemented on top of object oriented frameworks like CORBA and JavaRMI and may not be suitable for high performance computing according to a previous study [17].

In this paper, we propose a Grid-based RPC system called GrADSolve[1] that enables the users to invoke MPI applications on remote Grid resources from a sequential environment. In addition to providing easy-to-use interfaces for the service providers to upload the parallel applications into the system and for the end users to remotely invoke the parallel applications, GrADSolve performs application-level scheduling and dynamically choose the resources for the execution of the parallel applications based on the load dynamics of the Grid resources. GrADSolve also uses data distribution information provided by the library writers to partition the users' data and stage to the different resources used for the application execution. Our experiments show that the data staging mechanisms in GrADSolve helps reduce the data staging times in RPC systems by 20-50%. GrADSolve uses the popular Grid computing tool, Globus [8] for transferring data between the user and the end resources and for launching the application on the Grid resources. In addition to the above features, GrADSolve also enables the users to store execution traces for a problem run and use the execution traces for the subsequent problem runs.

Thus, the contributions of our research are:

1. design and development of an RPC system that utilizes standard Grid Computing mechanisms for invocation of remote parallel applications from a sequential environment.
2. selection of resources for parallel application execution based on load conditions of the resources and application characteristics.
3. communication of data between the user's address space and the Grid resources based on the data distribution used in the application and
4. maintenance of execution traces for problem runs.

Section 2 presents a detailed description of the framework of the GrADSolve system. The support in the GrADSolve system for maintaining execution traces is explained in Section 3. In Section 4, the experiments conducted in GrADSolve are explained and results are presented to demonstrate the usefulness of the data staging mechanisms and execution traces in GrADSolve. Section 5 looks at the related efforts in the development of RPC systems. Section 6 presents conclusions and future work.

2 Overview of GrADSolve

Figure 1 illustrates the overview of the GrADSolve system.

---

[1] The system is called GrADSolve since it is derived from the experiences of the GrADS [1] and NetSolve [3] projects.
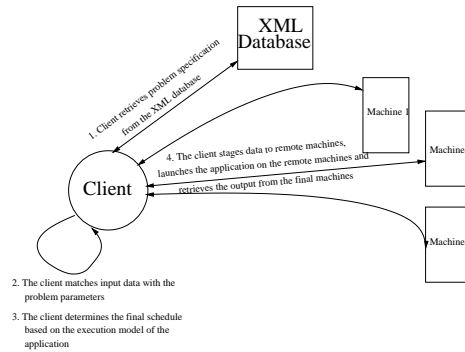
Figure 1. Overview of GrADSolve system

At the core of the GrADSolve system is a XML database implemented with Apache Xindice [19]. This database maintains four kinds of tables - *users*, *resources*, *applications* and *problems*. The *users*, *resources* and the *applications* tables contain information about the different users, machines and applications on the Grid system respectively. The *problems* table maintains information about the individual problem runs due to the invocation of the remote applications by the end users. The Xindice implementation of the XML-RPC standard [20] was used for storing and retrieving information to and from the XML database.

There are three human entities involved in GrADSolve - *administrators*, *library writers* and *end users*. The role of these entities in GrADSolve and the functions performed by the GrADSolve system for these entities are explained in the following sub sections.

2.1 Administrators

The GrADSolve administrator is responsible for managing the users and resources of the GrADSolve system. The administrator initializes the XML database and creates entities for different users and resources in the XML database by specifying configuration files that contains information for different users and resources, namely the user account names for different resources, the location of the home directories on different resources in the GrADSolve system, the names of the different machines, their computational capacities, the number of processors in the machines and other machine specifications.

2.2 Library Writers

The library writer uploads his application into the GrADSolve system by specifying an Interface Definition Language (IDL) file for the application. In the IDL file, the library writer specifies the programming language in which the function is written, the name of the function, the set of input and output arguments in the function, the description of the function, the names of the object files and

libraries needed for linking the function with other functions, if the function is sequential or parallel etc. For each input and output arguments, the library writer specifies the name of the argument, if the argument is an input or output argument, the datatype of the argument, the number of elements of the argument if the argument is a vector, the number of rows and columns of the argument if the argument is a matrix etc.

After the library writer submits the IDL file to the GrADSolve system, GrADSolve translates the IDL file to a XML document. The GrADSolve translation system also generates a wrapper program which acts as an entry point for remote execution of the actual function. After generating the wrapper program, the GrADSolve system compiles the wrapper program with the appropriate libraries and stages the executable file is to the remote machines in the GrADSolve system. Also, stored in the XML database for the application is the information regarding the location of the executable files for the application on the remote resources.

If the library writer wants to add an execution model for his application, he executes the *getperfmodel_template* utility specifying the name of the application. The utility retrieves the problem description of the application from the XML database and generates a performance model template file. The template file contains the definitions of the execution model routines. The library writer fills in the execution model routines with the appropriate code for specifying if the given set of resources have adequate capacity to solve the problem, the predicted execution cost of the application and the data distribution used in the application. The library writer uploads his execution model by executing the *add_perfmodel* utility. The *add_perfmodel* utility uploads the execution model for the application by storing the location of the wrapper program and the execution model to the XML database corresponding to the entry for the application.

2.3 End Users

The end users solve problems over remote GrADSolve resources by writing a client program. This client program can be written in C or Fortran. The client program includes an invocation of a routine called *gradsolve()* passing to the function, the name of the end application and the input and output parameters needed by the end application.

The invocation of the *gradSolve()* routine triggers the execution of the GrADSolve Application Manager. As a first step, the Application Manager verifies if the user has credentials to execute applications on the GrADSolve system. GrADSolve uses Globus Grid Security Infrastructure (GSI) for the authentication of users. The Application Manager then retrieves the problem description from the XML database and matches the user's data with the input and output parameters required by the end application.

If an execution model exists for the end application, the Application Manager downloads the execution model from the remote location where the library writer had previously stored the execution model. The Application Manager then retrieves the list of machines in the GrADSolve system from the *resources*

table in the XML database, and retrieves resource characteristics of the machines from the NWS [18]. The Application Manager uses the list of resources with machine characteristics, the execution model and scheduling heuristics [11, 21] to determine a final schedule for application execution and stores the status of the problem run and the final schedule in the *problems* table of the XML database corresponding to the entry for the problem run.

The Application Manager then creates working directories on the remote machines of the final schedule for end application execution and enters the *Application Launching* phase. The Application Launcher stores the input data to files and stages these files to the corresponding remote machines chosen for application execution. If data distribution information for an input data does not exist, the Application Launcher stages the entire input data to all the machines involved in end application execution. If the information regarding data distribution for an input data exists, the Application Launcher stages only the appropriate portions of the data to the corresponding machines. This kind of selective data staging significantly reduces the time needed for the staging for entire data especially if large amount of data is involved. After the staging of input data, the Application Launcher launches the end application on the remote machines chosen for the final schedule using the Globus MPICH-G mechanism. The end application reads the input data that were previously staged by the Application Launcher, solves the problem and stores the output data to the corresponding files on the machines in the final schedule. If the end application finished execution, the Application Launcher copies the output data from the remote machines to the user's memory space. The staging in of the output data from the remote locations is a reverse operation of the staging out of the input data to the remote locations. The GrADSolve Application Manager finally returns success state to the user client program.

3   Execution Traces in GrADSolve - Storage, Management and Usage

One of the unique features in the GrADSolve system is the ability provided to the users to store and use execution traces of problem runs. There are many applications in which the outputs of the problem depend on the exact number and configuration of the machines used for problem solving. Ill-conditioned problems or unstable algorithms can give rise to vast changes in output results due to small changes in input conditions. For these kinds of applications, the user may desire to use the same input environment for all problem runs. Also, during testing of new numerical algorithms over the Grid, different groups working on the algorithm may want to ensure that same results are obtained when the algorithms are executed with same input data on the same configuration of resources.

To guarantee reproducibility of numerical results in the above situations, GrADSolve provides capability to the users to store *execution traces* of problem runs and use the execution traces during subsequent executions of the same problem with the same input data. For storing the execution trace of the current

problem run, the user executes his GrADSolve program with a configuration file called *input.config* that contains a TRACE_FLAG variable that is either 0 or 1. During the registration of the problem run with the XML database, the value of the TRACE_FLAG variable is stored. After the end application completes its execution and the output data are copied from the remote machines to the user's memory, the Application Manager removes the remote files containing the input data for the end application if TRACE_FLAG is 0. But if the TRACE_FLAG is set to 1, the Application Manager retains the input data used for the problem run in the remote machines. At the end of the problem run, the Application Manager generates an output configuration file that contains a key corresponding to the execution trace.

When the user wants to execute the problem with a previously stored execution trace, he executes his client program specifying the TRACE_KEY variable in the *input.config* file. The TRACE_KEY variable is set with the key that corresponds to the execution trace. During the Schedule Generation phase, the Application Manager, instead of generating a schedule for the execution of the end application, retrieves the schedule used for the previous problem run corresponding to the TRACE_KEY, from the *problems* table in the XML database . The Application Manager then checks if the capacities of the resources in the schedule at the time of trace generation are comparable to the current capacities of the resources. If the capacities are comparable, the Application Manager proceeds to the rest of the phases of its execution. During the Application Launching phase, the Application Manager, instead of staging the input data to remote working directories, copies the input data and the data distribution information, used in the previous problem run corresponding to the TRACE_KEY, to the remote working directories. Thus GrADSolve guarantees the use of the same execution environment used in the previous problem run for the current problem run, and hence guarantees reproducibility of numerical results.

To support the storage and use of execution traces in the GrADSolve system, two trigger functions are associated with the XML database. One trigger function called *trace_usage_trigger* updates the last usage time of an execution trace when the execution trace is used for a problem run. Another trigger function called *cleanup_trigger* is used for periodically deleting entries in the *problems* table of the XML database thereby maintaining the size of the *problems* table in the database. Problems executed with execution traces are maintained for longer durations in the database, thus providing greater opportunity for the usage of the execution traces for the subsequent problem runs.

4 Experiments and Results

The GrADS testbed consists of about 40 machines from University of Tennessee (UT), University of Illinois, Urbana-Champaign (UIUC) and University of California, San Diego (UCSD). For the sake of clarity, our experimental testbed consists of 4 machines:

– a 933 MHz Pentium III machine with 512 MBytes of memory located in UT,

– a 450 MHz Pentium II machine with 256 MBytes of memory located in UIUC and
– 2 450 MHz Pentium III machines with 256 MBytes of memory located in UCSD.

The 2 UCSD machines are connected to each other by 100 Mb switched Ethernet. Machines from different locations are connected by Internet. In the experiments, GrADSolve was used to remotely invoke ScaLAPACK QR factorization. Block cyclic distribution was used for the matrix, A and the right-hand side vector, B. GrADSolve was operated in 3 modes. In the first mode, the execution model did not contain information about the data distribution used in the ScaLAPACK driver. In this case, GrADSolve transported the entire data to each of the locations used for the execution of the end application. This mode of operation is practiced in RPC systems that do not support the information regarding data distribution. In the second mode, the execution model contained information about the data distribution used in the end application. In this case, GrADSolve transported only the appropriate portions of the data to the locations used for the execution of end application. In the third mode, GrADSolve was used with an execution trace corresponding to a previous run of the same problem. In this case, data is not staged from the user's address space to the remote machines, but temporary copies of the input data used in the previous run are made for the current problem run.
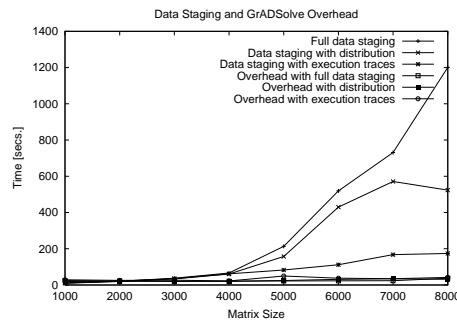

Figure 2. Data staging and other GrADSolve overhead

Figure 2 shows the times taken for data staging and other GrADSolve overhead for different matrix sizes and for the three modes of GrADSolve operation. The machines that were chosen by the GrADSolve application-level scheduler for the execution of end application for different matrix sizes are shown in Table 1. The decisions regarding the selction of machines for problem execution were automatically made by the GrADSolve system taking into account the size of the problems and the resource characteristics at the time of the experiments.

Comparing the first two modes in Figure 2, we find that for smaller problem sizes, the times taken for data staging in both the modes are the same. This is

Matrix size	Machines
1000	1 UT machine
2000	1 UT machine
3000	1 UT machine
4000	1 UT machine
5000	1 UT, 1 UIUC machines
6000	1 UIUC, 1 UCSD machines
7000	1 UIUC, 1 UCSD machines
8000	1 UT, 1 UIUC, 2 UCSD machines

Table 1. Machines chosen for application execution

because only one machine was used for problem execution and the same amount
of data are staged in both the modes when only one machine is involved for
problem execution. For larger problem sizes, the times for data staging with
distribution information is less than 20-55% of the times taken for staging the
entire data to remote resources. Thus the use of data distribution information in
GrADSolve can give significance performance benefits when compared to staging
the entire data that is practiced in some of the RPC systems. Data staging in
the third mode is basically the time taken for creating temporary copies of data
used in the previous problem runs in remote resources. We find this time to
be negligible when compared to the first two modes. Thus execution traces can
be used as caching mechanisms to use the previously staged data for problem
solving. The GrADSolve overheads for all the three modes are found to be the
same. This is because of the small number of machines used in the experiments.
For experiments when large number of machines are used, we predict that the
overheads will be higher in the first two modes than in the third mode. This
is because in the first two modes, the application-level scheduling will explore
large number of candidate schedules to determine the machines used for end
application while in the third mode, a previous application-level schedule will
be retrieved from the database and used.

5   Related Work

Few RPC systems contain mechanisms for the parallel execution of remote soft-
ware. The work by Maassen et. al [10] extends Java RMI for efficient communica-
tions in solving high performance computing problems. The framework requires
the end user's programs to be parallel programs. NetSolve [3] and Ninf [13]
support task parallelism by the asynchronous execution of number of remote
sequential applications. OmniRPC [14] is an extension of Ninf and supports
asynchronous RPC calls to be made from OpenMP programs. But similar to the
approaches in NetSolve, Ninf, and RCS, OmniRPC supports only master-worker
models of parallelism. NetSolve, Ninf and RCS also supports remote invocation
of MPI applications, but the amount of parallelism and the locations of the re-
sources to be used for the execution are fixed at the time when the applications

are uploaded to the systems and hence are not adaptive to dynamic loads in the Grid environments.

The efforts that are very closely related to GrADSolve are PaCO [12] and PaCO++ [6, 5] from the PARIS project in France. The PaCO systems are implemented within the CORBA [4] framework to encapsulate MPI applications in RPC systems. The data distribution and redistribution mechanisms in PaCO are much more robust than in GrADSolve and support invocation of remote parallel applications either from sequential or parallel client programs. Although the PARIS project aims to improve the performance of CORBA for high performance computing, the RPC mechanisms provided in CORBA by the use of client stubs and server skeletons have not found to be favorable for high performance computing according to a previous study [17]. Also, the PaCO projects do not support dynamic selection of resources for application execution as in GrADSolve. Also, GrADSolve supports Grid related security models by employing Globus mechanisms. And finally, GrADSolve is unique in maintaining execution traces that can help bypass the resource selection and data staging phases.

6 Conclusions and Future Work

In this paper, an RPC system for efficient execution of remote parallel software was discussed. The efficiency is achieved by dynamically choosing the machines used for parallel execution and staging the data to remote machines based on data distribution information. The GrADSolve RPC system also supports maintaining and utilizing execution traces for problem solving. Our experiments showed that the GrADSolve system is able to adapt to the problem sizes and the resource characteristics and yielded significant performance benefits with its data staging and execution trace mechanisms.

Interfaces to the library writers for expressing more capabilities of the end application are currently being designed. These capabilities include the ability of the application to be preempted and continued later with different processor configuration. These capabilities will allow GrADSolve to adapt to changing Grid scenarios. Remote execution of non-MPI parallel programs and applications with different modes of parallelism are also being considered. Support for remote invocation in different programming languages including MATLAB are also part of our future efforts.

References

1. F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Applications and Supercomputing*, 15(4):327–344, Winter 2001.
2. A.D. Birrell and B.J. Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

3. H. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, Fall 1997.

4. CORBA http://www.corba.org.

5. A. Denis, C. Pérez, and T. Priol. Achieving Portable and Efficient Parallel CORBA Objects. *Concurrency and Computation: Practice and Experience*, 2002.

6. A. Denis, C. Prez, and T. Priol. Portable Parallel CORBA Objects: an Approach to Combine Parallel and Distributed Programming for Grid Computing. In *Proc. of the 7th International Euro-Par'01 Conference (EuroPar'01)*, pages 835–844. Springer, August 2001.

7. I. Foster and C. Kesselman eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, ISBN 1-55860-475-8, 1999.

8. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.

9. Java Remote Method Invocation (Java RMI) java.sun.com/products/jdk/rmi.

10. J. Maassen, R. van Nieuwpoort, R. Veldema, H. Bal, T. Kielmann, C. Jacobs, and R. Hofman. Efficient Java RMI for Parallel Programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(6):747–775, November 2001.

11. A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, and S. Vadhiyar. Numerical Libraries and the Grid: The GrADS Experiments with Scalapack. *Journal of High Performance Applications and Supercomputing*, 15(4):359–374, Winter 2001.

12. C. René and T. Priol. MPI Code Encapsulating using Parallel CORBA Object. *Cluster Computing*, 3(4):255–263, 2000.

13. H. Nakada M. Sato and S. Sekiguchi. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metascomputing Issue*, 15(5-6):649–658, 1999.

14. M. Sato, M. Hirano, Y. Tanaka, and S. Sekiguchi. OmniRPC: A Grid RPC Facility for Cluster and Global Computing in OpenMP. In *In Workshop on OpenMP Applications and Tools (WOMPAT2001)*, July 2001.

15. K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova. Overview of gridrpc: A remote procedure call api for grid computing. In M. Parashar, editor, *Lecture notes in computer science 2536 Grid Computing - GRID 2002*, volume Third International Workshop, pages 274–278, Baltimore, MD, USA, November 2002. Springer Verlag.

16. Simple Object Access Protocol (SOAP) http://www.w3.org/TR/SOAP.

17. T. Suzumura, T. Nakagawa, S. Matsuoka, H. Nakada, and S. Sekiguchi. Are Global Computing Systems Useful? - Comparison of Client-Server Global Computing Systems Ninf, Netsolve versus CORBA. In *In Proceedings of the 14th International Parallel and Distributed Processing Symposium, IPDPS '00*, pages 547–559, May 2000.

18. R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, October 1999.

19. Apache Xindice http://xml.apache.org/xindice.

20. XML-RPC http://www.xmlrpc.com.

21. A. Yarkhan and J. Dongarra. Experiments with Scheduling Using Simulated Annealing in a Grid Environment. In M. Parashar, editor, *Lecture notes in computer science 2536 Grid Computing - GRID 2002*, volume Third International Workshop, pages 232–242, Baltimore, MD, USA, November 2002. Springer Verlag.