GrADSolve - a Grid-based RPC system for Parallel Computing with Application-Level Scheduling [⋆]

Sathish S. Vadhiyar [*],

Department of Computer Science, University of Tennessee

Jack J. Dongarra

Department of Computer Science, University of Tennessee and Oak Ridge National Laboratory

Abstract

Although some existing Remote Procedure Call (RPC) systems provide support for remote invocation of parallel applications, these RPC systems lack powerful scheduling methodologies for the dynamic selection of resources for the execution of parallel applications. Some RPC systems support parallel execution of software routines with simple modes of parallelism. Some RPC systems statically choose the configuration of resources for parallel execution even before the parallel routines are invoked remotely by the end user. These policies of the existing systems prevent them from being used for remotely solving computationally intensive parallel applications over dynamic computational Grid environments. 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 resources and the characteristics of the application. Application-level scheduling is employed for taking into account both the application and resource properties. GrADSolve also 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 the execution traces maintained by GrADSolve for problem solving.

Key words: RPC, Grid, GrADSolve, application-level scheduling, data staging, execution traces

1 Introduction

Remote Procedure Call (RPC) mechanisms have been studied extensively and have been found to be powerful abstractions for distributed computing [1,2]. In RPC frameworks, the end user invokes a simple routine to solve problems

⋆ This work is supported in part by the National Science Foundation contract GRANT #EIA-9975020, SC #R36505-29200099 and GRANT #EIA-9975015

* Corresponding author.
Email addresses: vss@cs.utk.edu (Sathish S. Vadhiyar), dongarra@cs.utk.edu (Jack J. Dongarra).

over remote distributed resources. A number of RPC frameworks have been implemented and are widely used [3–11]. In addition to providing simple interfaces for uploading applications into the distributed systems and for remote invocation of the applications, some of the RPC systems also provide service discovery, resource management, scheduling, security and information services.

The role of RPC in Computational Grids [12] has been the subject of many recent studies [13–17]. Computational Grids consist of large number of machines ranging from workstations to supercomputers and strive to provide transparency to end users and high performance for end applications. While high performance is achieved by the parallel execution of applications on a large number of Grid resources, user transparency can be achieving by employing RPC mechanisms. Hence, Grid-based RPC systems need to be built to provide the end users the capability to invoke remote parallel applications on Grid resources using a simple sequential procedure call.

Though there are a large number of RPC systems that support remote invocation of parallel applications [7,9,10,17–20], the selection of resources for the execution of parallel applications in these systems does not take into account the dynamic load aspects that are associated with Computational Grids. Some of the parallel RPC systems [21–26,18–20] are mainly concerned with providing robust and efficient interfaces for the service providers to integrate their parallel applications into the systems, and for the end users to remotely use these parallel services. In these systems the users or the service providers have to provide their own scheduling mechanisms if the resources for end application execution have to be dynamically chosen. In most cases, the users and the service providers lack the expertise to implement scheduling techniques. Some RPC systems [9,10] provide scheduling services in addition to the basic functionality of providing interfaces to the service providers and users. In these systems, scheduling methodologies are employed to choose between different parallel domains that implement the same parallel services. But within a parallel domain, the number and configuration of resources are fixed 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.

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. GrADSolve combines the easy-to-use RPC mechanisms of NetSolve [9,28] and powerful application-level scheduling mechanisms inherent in the GrADS [27] project. Application-level scheduling has been proven to be a powerful scheduling technique for providing high performance [29,30].

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 also provides interfaces

[1] The system is called GrADSolve since it is derived from the experiences of the GrADS [27] and NetSolve [9,28] projects.

for the service providers or library writers to upload execution models that provide information about the predicted execution costs of the applications. This information is used by GrADSolve to perform application-level scheduling and to dynamically choose the resources for the execution of the parallel applications based on the load dynamics of the Grid resources. GrADSolve also uses the data distribution information provided by the library writers to partition the users' data and stage the data to the different resources used for the application execution. Our experiments show that the data staging mechanisms in GrADSolve help reduce the data staging times in RPC systems by 20-50%. GrADSolve also uses the popular Grid computing tool, Globus [31] 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. This feature helps in significantly reducing the overhead incurred due to the selection of the resources for application execution and the staging of input data to the end resources.

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) maintenance of execution traces for problem runs.

Section 2 describes in brief the GrADS and NetSolve projects. The architecture of GrADSolve, the various entities in the GrADSolve system and the support for the entities in the GrADSolve system are explained in Section 3. The support in the GrADSolve system for maintaining execution traces is explained in Section 4. In Section 5, 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 6 looks at the related efforts in the development of parallel RPC systems. Section 7 presents conclusions and future work.

2 Background of GrADSolve

GrADSolve evolved from two projects, GrADS [27] and NetSolve [9]. In this section, the overviews of GrADS and NetSolve are presented.

2.1 The GrADS project

The Grid Application Development Software (GrADS) project is a multi-university research project which works to simplify distributed heterogeneous computing in the same way that the World Wide Web simplified informa-

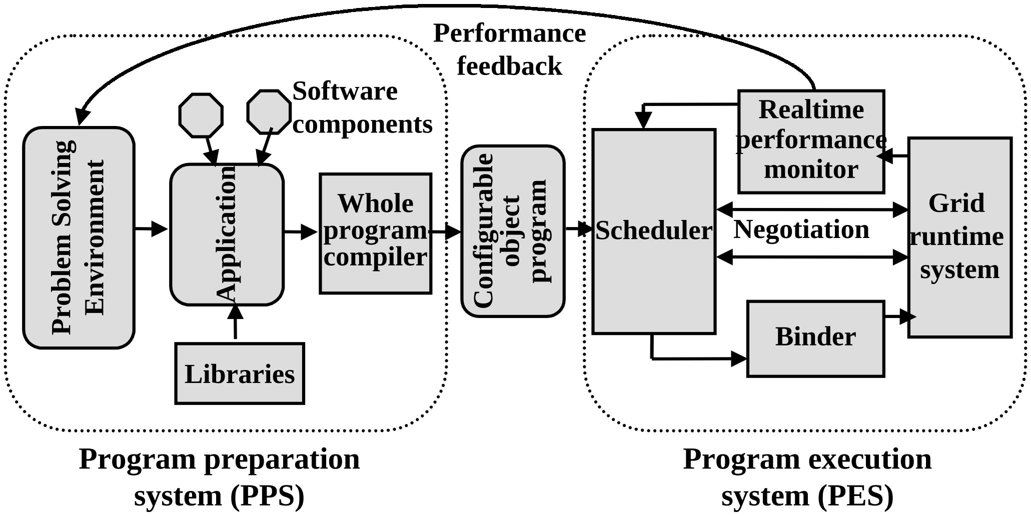
Performance feedback
Software components
Problem Solving Environment
Application
Whole program compiler
Libraries
Configurable object program
Scheduler
Realtime performance monitor
Negotiation
Grid runtime system
Binder
Program preparation system (PPS)
Program execution system (PES)

Fig. 1. The GrADS Architecture

tion sharing. The GrADS project intends to provide tools and technologies for the development and execution of applications in a Grid environment. In the GrADS vision, the end user simply presents their parallel application to the framework for execution. The framework is responsible for scheduling the application on an appropriate set of resources, launching and monitoring the execution, and, if necessary, rescheduling the application on a different set of resources. A high-level view of the GrADS architecture is shown in Figure 1. For more details, the readers are referred to [27].

2.2 NetSolve - A Brief Overview

NetSolve [9] is a Grid computing system developed at University of Tennessee. It is a Remote Procedure Call (RPC) based system used for solving numerical applications over remote machines. The NetSolve system consists of 3 main components - agent, server and client. The working of the NetSolve system

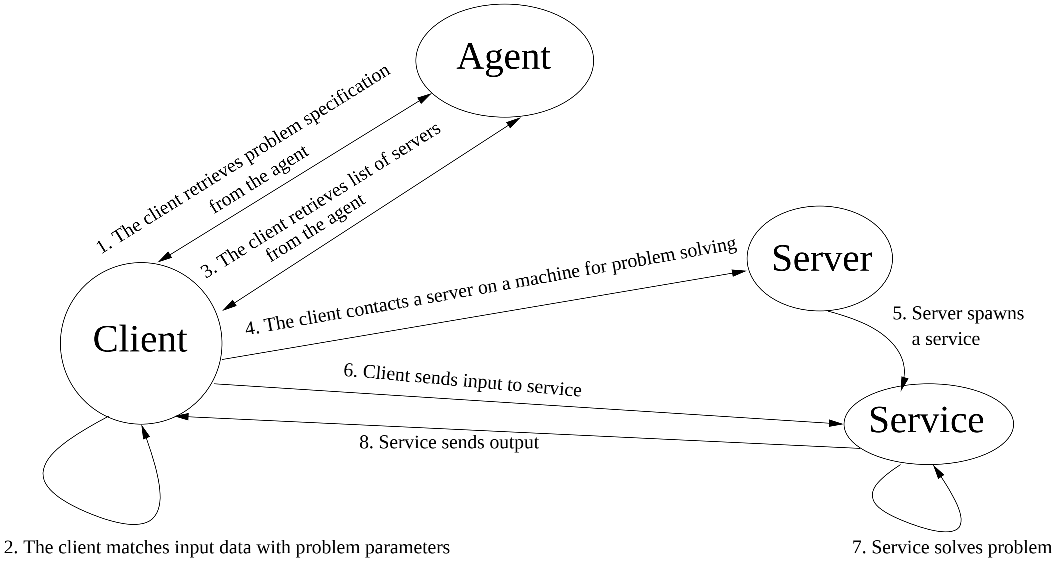
Agent
Client
Server
Service
1. The client retrieves problem specification from the agent
2. The client matches input data with problem parameters
3. The client retrieves list of servers from the agent
4. The client contacts a server on a machine for problem solving
5. Server spawns a service
6. Client sends input to service
7. Service solves problem
8. Service sends output

Fig. 2. Overview of NetSolve System

is illustrated in Figure 2. Although NetSolve supports remote execution of parallel applications, the amount of parallelism is fixed at the time the server daemons are started. For more details, the readers are referred to [9].

3 The GrADSolve System

The general architecture of GrADSolve is shown in Figure 3. At the core of the GrADSolve system is a XML database implemented with Apache Xindice [32]. GrADSolve uses XML as a language for storing information about different Grid entities. This database maintains four kinds of tables - *users*, *resources*, *applications* and *problems*. The Xindice implementation of the XML-RPC standard [4] 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

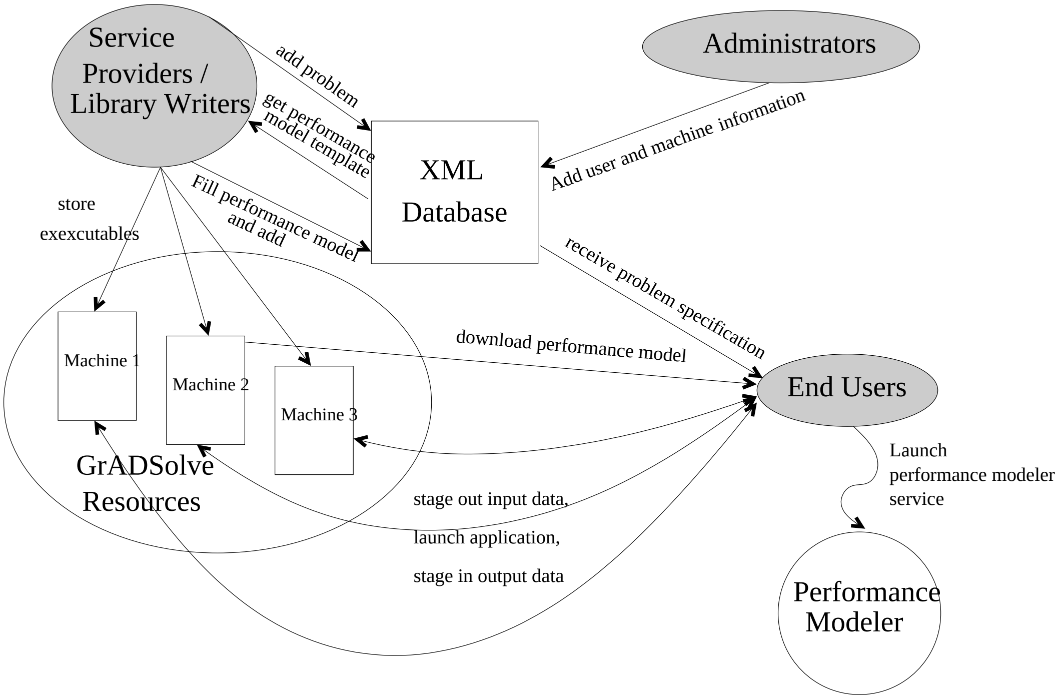
Service Providers / Library Writers
add problem
get performance model template
Fill performance model and add
store exexcutables
XML Database
Administrators
Add user and machine information
receive problem specification
Machine 1
Machine 2
Machine 3
GrADSolve Resources
download performance model
End Users
stage out input data,
launch application,
stage in output data
Launch performance modeler service
Performance Modeler

Fig. 3. Overview of GrADSolve system

GrADSolve and the functions performed by the GrADSolve system for supporting these entities are explained in the following subsections.

3.1 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 in the XML database by specifying a *user configuration file*. The user configuration file contains information for different users, namely the user account names for different resources and the location of the home directories on different resources in the GrADSolve system. The administrator also creates the *resources* table in the Xindice database and adds entries for different resources in the GrADSolve system by specifying a *resource configuration file*. The various information in the configuration file, namely the names of the different machines, their computational capacities, the number of processors in the machines and other machine specifications, are stored as XML documents.

3.2 Library Writers

The library writer uploads his application into the GrADSolve system by specifying an Interface Definition Language (IDL) file for the application. Even though there are a number of robust IDL systems for High Performance Computing including the OMG IDL [33] and the SIDL (Scientific IDL) from the BABEL project [34], GrADSolve implements and uses its own IDL system. In addition to the primitives for MPI applications and complex data types that are supported by the existing IDL systems, GrADSolve supports sparse matrix data types. Supporting sparse matrix data types is essential for integrating popular and efficient high performance libraries including PETSC [35–37], AZTEC [38], SuperLU [39] etc. Figure 4 illustrates an IDL with support for sparse matrices that is unique to GrADSolve IDL. In the IDL file, the 3rd parameter, *SM*, is a sparse matrix represented by a compressed-row format.

After the library writer submits the IDL file to the GrADSolve system, GrADSolve translates the IDL file to a XML document similar to the mechanisms in

```
PROBLEM sparse_solve
C ROUTINE sparse_solve(IN string package,
                       IN int N,
                       IN double SM[N][N]<nnz,indices,pointer>,
                       IN int nnz,
                       IN int indices[nnz],
                       IN int pointer[N],
                       IN double rhs[N],
                       IN double pivot,
                       IN int permutation,
                       OUT double sol[N])
TYPE = parallel
```

Fig. 4. An example GrADSolve IDL for a sparse factorization problem

SIDL. The GrADSolve translation system also generates a wrapper program that acts as an entry point for remote execution of the actual function. The wrapper program performs initialization of the parallel environment, reads input data from files, invokes the actual parallel routine and stores output data to files. The GrADSolve system then compiles the wrapper program with the object files and the libraries specified in the IDL file and with the appropriate parallel libraries if the application is specified as a parallel application in the IDL file. The GrADSolve system then stages the executable to the different resources in the Grid using the Globus GridFTP mechanisms and stores the locations of the executables in the XML database.

The library writer also has the option of adding an execution model for the application. If the library writer wants to add an execution model, 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 performance model template file contains definitions for three functions to help the library writers to convey information about his library to the GrADSolve system - *areResourcesSufficient* for conveying if a given set of resources are adequate for problem solving, *getExecutionTimeCost* for conveying the predicted execution cost of the application if executed on a given set of resources and an optional function *mapper* for specifying the data distribution of the different data used by the application. The performance model template file generated by the *getperfmodel_template* for a ScaLAPACK QR problem is shown in Figure 5. 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 execution model in the XML database corresponding to the entry for the application.

3.3 End Users

The end users solve problems over remote GrADSolve resources by writing a client program 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.

```
int areResourcesSufficient(int N, int NB, double* A,
                           double* B,
                           RESOURCEINFO* resourceInfo,
                           SCHEDULESTRUCT* schedule){

}

int getExecutionTimeCost(int N, int NB, double* A,
                         double* B,
                         RESOURCEINFO* resourceInfo,
                         SCHEDULESTRUCT* schedule,
                         double* cost{

}

int mapper(int N, int NB, double* A, double* B,
           RESOURCEINFO* resourceInfo,
           SCHEDULESTRUCT* inputSchedule,
           SCHEDULESTRUCT* mapperSchedule){


}
```

Fig. 5. A Performance Model template generated by the GrADSolve system for the ScaLAPACK QR problem

GrADSolve uses Globus Grid Security Infrastructure (GSI) [40] for the authentication of users. If the application exists in the GrADSolve system, the Application Manager registers the problem run in the *problems* table of the XML database. 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 compiles the execution model programs with algorithms for scheduling [41,42] and starts the application-specific *Performance Modeler* service. The Application Manager then retrieves the list of machines in the GrADSolve system from the *resources* table in the XML database, and retrieves various performance characteristics of the machines including the peak performance of the resources, the load on the machines, the latency and the bandwidth of the networks between the machines and the free memory available on the machines from the Network Weather Service (NWS) [43]. The Application Manager passes the list of machines, along with the resource characteristics to the Performance Modeler service to determine if the resources are sufficient to solve the problem. If the resources are sufficient, the Application Manager proceeds to the *Schedule Generation* phase.

In the Schedule Generation phase, the Application Manager first determines if the end application has an execution model. If an execution model exists, the Application Manager contacts the Performance Modeler service and passes the problem parameters and the list of machines with the machine capabilities. The Performance Modeler service uses the execution model supplied by the library writer along with certain scheduling heuristics [42,41] to determine a final schedule for application execution and returns the final list of machines to the Application Manager. Along with the final list of machines and the predicted execution cost for the final schedule, the Performance Modeling service also returns information about the data distribution for the different data in the end application. If an execution model does not exist for the end application, the Schedule Generation phase adopts default scheduling strategies to generate the final schedule for end application execution. At the end of the Schedule Generation phase, the GrADSolve Application Manager receives a list of machines for final application execution. The Application Manager then 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 Launching phase consists of several important functions. The Application Launcher stores the input data to files and stages these files to the corresponding remote machines chosen for application execution using the Globus GridFTP mechanisms. 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 the 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 a 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 [44] mechanism. The end application reads the input data that were previously staged by the Application Launcher and solves the problem. The end application then 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 address 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.

4 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. For example, considering the problem of adding a large number of double precision numbers, one of the parallel implementations of the problem is to partition the list of double precision numbers among all processes of the parallel application, compute local sums of the numbers in each process and then compute the global sum of the local sums computed on each process. The final sum obtained for the same set of double precision numbers may vary from one problem run to another depending on the number of elements in each partition, the number of processes used in the parallel application and the actual processors used in the computation. This is due to the impact of the round off errors caused by the addition of double precision numbers. In general, 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 the testing of new numerical algorithms over the Grid, different groups working on the algorithm may want to ensure that the 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 an execution trace of the current problem run, the user executes his GrADSolve program with a configuration file called *input.config* in the working directory containing the line, TRACE_FLAG = 1.

During the registration of the problem run with the XML database, the value of the TRACE_FLAG variable is stored. The GrADSolve Application Manager proceeds to other stages of its execution. After the end application completes its execution and the output data are copied from the remote machines to the user's address space, the Application Manager, under default mode of operation, removes the remote working directories used for storing the files containing the input data for the end application. But when the user wants to store the execution trace of the problem run, i.e. when the *input.config* file contains "TRACE_FLAG = 1" line, 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 called *output.config* containing the line, TRACE_KEY = <key>. The value *key* in the *output.config* is a pointer to the execution trace stored for the problem run.

When the user wants to execute the problem with the execution trace previously stored, he executes his client program specifying the line, TRACE_KEY = <key> in the *input.config* file. The value *key* in the input.config, is the same value previously generated by the GrADSolve Application Manager when the execution trace was stored. The Application Manager first checks if the TRACE_KEY exists in the *problems* table of the XML database. If the TRACE_KEY does not exist, the Application Manager displays an error message to the user and aborts operation. If the TRACE_KEY exists for an execution trace of a previous problem run, the Application Manager registers the current problem run with the XML database and proceeds to the other stages of its execution. 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 not comparable, the Application Manager displays an error message to the user and aborts the operation. 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. The use of the same number of machines and the same input data used in the previous schedule also guarantees the use of the same data distribution for the current problem run. 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. The *cleanup_trigger* is invoked whenever a new entry corresponding to a problem run is added to the *problems* table. The *cleanup_trigger* employs a longer duration for those problem runs for which execution traces were stored.

5 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 the ScaLAPACK driver for solving a linear system of equations, $AX = B$. The driver invokes ScaLAPACK QR factorization for the factorization of matrix, A. Block-cyclic distribution was used for the matrix A. A GrADSolve IDL was written for the driver routine and an execution model that predicts the execution cost of the QR problem was uploaded into the GrADSolve system. The GrADSolve user invokes the remote parallel application by passing the size of the matrix A and the right-hand side vector, B to the *gradsolve()* call.

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 6 shows the times taken for data staging and other GrADSolve overhead for different matrix sizes and for the three modes of GrADSolve operation.

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

Since the times taken for the execution of the end application are same in all the three modes, we focus only on the times taken for data staging and possible Grid overheads. 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 UT machine was used for smaller problem sizes since it had larger computing power than other machines. For matrix size, 5000, a UIUC machine was also used for the execution of parallel application. For matrix sizes, 6000 and 7000, the available memory in the UT machine at the time of the experiments was less than the memory needed for the problems. Hence UIUC and UCSD machines were used. For matrix size, 8000, all 4 machines were needed to accommodate the problem. All the above decisions 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.

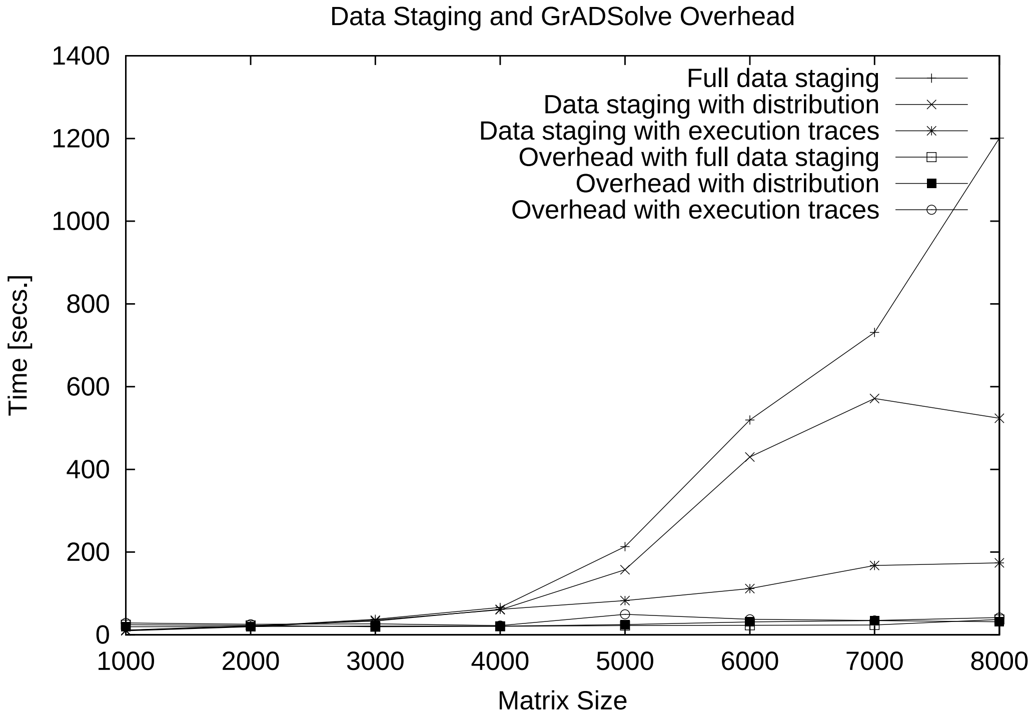

Fig. 6. Data staging and other GrADSolve overhead

Comparing the first two modes in Figure 6, we find that for smaller problem sizes, the times taken for data staging in both the modes are the same. This is 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.

6 Related Work

A number of parallel RPC systems have been built in the context of Object Management Group (OMG) [21], Common Component Architecture Forum (CCA) [22] and Grid research efforts [9,10,13].

The Object Management Group (OMG) [21] has been dealing with specifying objects for both sequential and parallel applications. The Data Parallel CORBA specification describes parallel objects that enable the object implementer to take advantage of parallel resources for achieving high performance. The specification defines interfaces for both the implementers of the objects and the client to use the remote parallel services. The specification does not deal with dynamic selection of resources for parallel computing. The PaCO [19,45] and PaCO++ [14,20] systems from the PARIS project in France are implemented within the CORBA [5] 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. Recently, the PARIS project has been investigating coupling multiple applications of different types in Grid frameworks [15,16]. Similar to the Data Parallel CORBA specification, the parallel CORBA objects in the PaCO projects do not support dynamic selection of resources for application execution as in GrADSolve. The selection of resources for parallel execution taking into account the load aspects of the resources is a necessity in dynamic Computational Grids. 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.

The Common Component Architecture Forum (CCA) [22] has been investigating the deployment and use of both parallel and sequential components. Their “MxN Redistribution” working group has been dealing with the issues of data redistribution when multiple parallel components are coupled together. The CUMULVS MxN interface [23] from Oak Ridge National Laboratory, the PAWS environment [24] from Los Alamos National Laboratory and the

PARDIS SPMD objects [25] from Indiana University work within the CCA to develop a parallel RPC standard. The main goals of these systems include providing interoperability between different components, building user interfaces for conveying information about the parallel data, developing communication schedules to communicate the data between different components and synchronizing data transfers. These projects delegate the responsibility of scheduling or choosing the end resources for parallel application to the implementers of parallel components. In most cases, the implementers of the components lack the expertise to include scheduling technologies. In GrADSolve, application-level scheduling is an integral component of the system and requires the implementors of the parallel components to only provide information about their parallel applications.

NetSolve [9] and Ninf [10] are Grid computing systems that support task parallelism by the asynchronous execution of a number of remote sequential applications. OmniRPC [17] is an extension of Ninf and supports asynchronous RPC calls to be made from OpenMP programs. But similar to the approaches in NetSolve and Ninf, OmniRPC supports only master-worker models of parallelism. NetSolve and Ninf also support remote invocation of MPI applications, but the amount of parallelism and the locations of the resources 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. Recently, Grid-RPC [13] has been proposed to standardize the efforts of NetSolve and Ninf. The current Grid-RPC standard does not specify scheduling methodologies to choose the resources for execution of remote parallel applications.

7 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 for 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 resumed later with different processor configuration. These capabilities will allow GrADSolve to adapt to changing Grid scenarios. The current GrADSolve system employs application-level scheduling requiring the implementers to provide information about their libraries. In the future, we plan to employ methods for automatic determination of the information about the libraries similar to the efforts in the Prophesy [46] project. Though GrADSolve currently provides basic security by the authentication of users and service providers through Globus mechanisms, it does not provide privacy with regard to message transactions and also does not support validation of results. We plan to employ encryption of data to provide privacy and signed software mechanisms to assure integrity of results.

8 Acknowledgments

The authors would like to thank the managers of the GrADS project for providing valuable input during the development of GrADSolve. We acknowledge the use of machines in the GrADS testbed for the experiments conducted in the research. We also thank the research teams from different institutions, namely the Pablo research group from the University of Illinois, Urbana-Champaign, the Grid Research and Innovation Laboratory (GRAIL) from the University of California, San Diego and the Innovative Computing Laboratory (ICL) from University of Tennessee, for the support and maintenance of the machines in the GrADS testbed and for enabling of experiments needed for the research.

References

[1] A. Birrell, B. Nelson, Implementing Remote Procedure Calls, ACM Transactions on Computer Systems 2 (1) (1984) 39–59.

[2] B. Bershad, T. Anderson, E. Lazowska, H. Levy, Lightweight Remote Procedure

Call, ACM Transactions on Computer Systems (TOCS) 8 (1) (1990) 37–55.

[3] Simple Object Access Protocol (SOAP) http://www.w3.org/TR/SOAP.

[4] XML-RPC http://www.xmlrpc.com.

[5] CORBA http://www.corba.org.

[6] Java Remote Method Invocation (Java RMI) java.sun.com/products/jdk/rmi.

[7] C.-C. Chang, G. Czajkowski, T. von Eicken, MRPC: A High Performance RPC System for MPMD Parallel Computing 29 (1) (1999) 43–66.

[8] R. Rabenseifner, The DFN Remote Procedure Call Tool for Parallel and Distributed Applications, in: In Kommunikation in Verteilten Systemen - KiVS 95. K. Franke, U. Huebner, W. Kalfa (Editors), Proceedings, Chemnitz-Zwickau, 1995, pp. 415–419.

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

[10] H. N. M. Sato, S. Sekiguchi, Design and Implementations of Ninf: Towards a Global Computing Infrastructure, Future Generation Computing Systems, Metascomputing Issue 15 (5-6) (1999) 649–658.

[11] P. Arbenz, W. Gander, M. Oettli, The Remote Computation System, Parallel Computing 23 (1997) 1421–1428.

[12] I. Foster, C. K. eds., The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, ISBN 1-55860-475-8, 1999.

[13] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, H. Casanova, Overview of GridRPC: A Remote Procedure Call API for Grid Computing, in: M. Parashar (Ed.), Lecture notes in computer science 2536 Grid Computing - GRID 2002, Vol. Third International Workshop, Springer Verlag, Baltimore, MD, USA, 2002, pp. 274–278.

[14] A. Denis, C. Prez, 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), Springer, 2001, pp. 835–844.

[15] A. Denis, C. Prez, . Priol, Towards High Performance CORBA and MPI Middlewares for Grid Computing, in: C. A. Lee (Ed.), Proc. of the 2nd International Workshop on Grid Computing, no. 2242 in LNCS, Springer-Verlag, 2001, pp. 14–25.

[16] C. Prez, T. Priol, A. Ribes, A Parallel CORBA Component Model for Numerical Code Coupling, in: C. A. Lee (Ed.), Proc. of the 3nd International Workshop on Grid Computing, LNCS, Springer-Verlag, 2002.

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

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

[19] C. René, T. Priol, MPI Code Encapsulation using Parallel CORBA Object, in: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing, IEEE, 1999, pp. 3–10.

[20] A. Denis, C. Pérez, T. Priol, Achieving Portable and Efficient Parallel CORBA Objects, Concurrency and Computation: Practice and Experience .

[21] Object Management Group `http://www.omg.org`.

[22] Common Component Architecture `http://www.cca-forum.org`.

[23] G. A. Geist, J. A. Kohl, P. M. Papadopoulos, CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications, International Journal of High Performance Computing Applications 11 (3) (1997) 224–236.

[24] P. H. Beckman, P. K. Fasel, W. F. Humphrey, S. M. Mniszewski, Efficient Coupling of Parallel Applications Using PAWS, in: Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing, IEEE, 1998, pp. 215–223.

[25] K. Keahey, D. Gannon, PARDIS: A Parallel Approach to CORBA, in: Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, IEEE, 1997, pp. 31–39.

[26] A. Denis, C. Pérez, T. Priol, PadicoTM: An Open Integration Framework for Communication Middleware and Runtimes, Future Generation Computer Systems 19 (2003) 575–585.

[27] 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, R. Wolski, The GrADS Project: Software Support for High-Level Grid Application Development, International Journal of High Performance Applications and Supercomputing 15 (4) (2001) 327–344.

[28] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, S. Vadhiyar, Users' Guide to NetSolve V1.4.1, Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN (June 2002).

[29] F. Berman, High-performance Schedulers, in: The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, ISBN 1-55860-475-8, 1999, pp. 279–203.

[30] F. Berman, R. Wolski, The AppLeS Project: A Status Report, Proceedings of the 8th NEC Research Symposium .

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

[32] Apache Xindice `http://xml.apache.org/xindice`.

[33] OMG IDL `http://www.omg.org/gettingstarted/omg_idl.htm`.

[34] SIDL from BABEL project `http://www.llnl.gov/CASC/components/babel.html`.

[35] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A. M. Bruaset, H. P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhauser Press, 1997, pp. 163–202.

[36] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith, PETSc home page, http://www.mcs.anl.gov/petsc (1999).

[37] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith, PETSc 2.0 users manual, Tech. Rep. ANL-95/11 - Revision 2.0.24, Argonne National Laboratory (1999).

[38] AZTEC `http://www.cs.sandia.gov/CRF/aztec1.html`.

[39] SuperLU `http://crd.lbl.gov/∼xiaoye/SuperLU`.

[40] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch, A National-Scale Authentication Infrastructure, IEEE Computer 33 (12) (2000) 60–66.

[41] A. Yarkhan, J. Dongarra, Experiments with Scheduling Using Simulated Annealing in a Grid Environment, in: M. Parashar (Ed.), Lecture notes in Computer Science 2536 Grid Computing - GRID 2002, Vol. Third International Workshop, Springer Verlag, Baltimore, MD, USA, 2002, pp. 232–242.

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

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

[44] I. Foster, N. Karonis, A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems, In Proceedings of SuperComputing 98 (SC98)

.

[45] C. René, T. Priol, MPI Code Encapsulating using Parallel CORBA Object, Cluster Computing 3 (4) (2000) 255–263.

[46] V. E. Taylor, X. Wu, J. Geisler, R. Stevens, Using Kernel Couplings to Predict Parallel Application Performance, in: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, IEEE, 2002, pp. 125–135.