A Grid-Aware Web Interface with Advanced Service Trading for Linear Algebra Calculations

Hrachya Astsatryan[1], Vladimir Sahakyan[1], Yuri Shoukouryan[1],
Michel Daydé[2], Aurelie Hurault[2], Marc Pantel[2], Eddy Caron[3],

[1] Institute for Informatics and Automation Problems of the National Academy of Sciences of the Republic of Armenia, 1, P. Sevak str., Yerevan, 0014, Armenia
{hrach, svlad, shouk}@sci.am

[2] Institut de Recherche en Informatique de Toulouse, Ecole Nationale Supérieure d´Electrotechnique, d´Electronique, d´Informatique, d´Hydraulique et des Télécommunications, 2, rue Charles Camichel, 31071, Toulouse, France
{Michel.Dayde,Aurelie.Hurault,Marc.Pantel}@enseeiht.fr

[3] Laboratoire de l'Informatique du Parallélisme de Ecoles Normales Supérieure de Lyon, 46, Allée d'Italie, 69364, Lyon, France
Eddy.Caron@ens-lyon.fr

Abstract. Due to the rapid growth of the Internet and Web, there has been a rising interest in using the Web as an interface to develop various applications over computational Grid environments. The purpose of this work is to develop a Grid-aware Web interface for linear algebra tasks with advanced service trading. Developing efficient and portable codes, requires users to face parallel computing and programming and to make use of different standard libraries, such as the Blas [1], Lapack [2] and Scalapack [3] in order to solve computational tasks related to linear algebra. For this purpose, a scientific computing environment based on a Web interface is described that allows users to perform their linear algebra tasks without explicitly calling the above mentioned libraries and software tools, as well as without installing any piece of software on local computers: users enter algebraic formula (such as in Matlab or Scilab [4]) that are evaluated for determining the combinations of services answering the user request. Services are then executed locally or over the Grid using the Distributed Interactive Engineering Toolbox (DIET) [5] middleware.

Keywords: Grid computing, Linear Algebra, Scientific Computing, Web Interface, Service Trading.

1 Introduction

Scientific computing aims at constructing mathematical models and numerical solution techniques for solving problems arising in science (including life and social sciences) and engineering. The solution of linear system of equations lies at the heart of most calculations in scientific computing. For the past twenty years, there has been a great deal of activity in the area of algorithms and software for solving linear algebra problems. Scalapack is a library of high-performance linear algebra routines for distributed-memory message-passing MIMD computers and networks of workstations supporting PVM and/or MPI. It contains routines for solving systems of linear equations, least squares problems and eigenvalue problems. The Scalapack library is built upon a set of communication routines which are themselves a library, called the "BLACS" (Basic Linear Algebra Communication Subprograms) [6]. Furthermore, with Scalapack comes an additional library called the "PBLAS", which can be seen as a parallel distributed version of the "BLAS" (Basic Linear Algebra Subprograms). Using the PBLAS simple matrix/vector and more complex operations can be performed in parallel.

Numerical simulations involving massive amounts of calculations are often executed on supercomputers or distributed computing platforms (e.g. clusters and Grids). It is often difficult for non expert users to use such computational infrastructures or even to use advanced libraries over clusters or Grids, because they often require to be familiar with Linux base OS, Grid computing middleware and tools (Glite, Globus, Condor; PBS, OAR; JDL, etc..), parallel programming techniques and packages (MPI; Mpich, Lam MPI; etc..) and linear algebra libraries (Scalapack, Lapack, Blas, etc..). In order to overcome the above mentioned problems for linear algebra tasks, a Web interface has been developed, which is continuation of previous work [7].

The purpose of our work is to develop a Grid-aware Web interface for linear algebra tasks with advanced service trading. It provides a seamless bridge between high performance linear algebra libraries such as Scalapack and those users who are not familiar with linear algebra softwares. Note that even when considering a well-defined area such as linear algebra, determining the combination of functions (services) of BLAS, Lapack and Scalapack libraries satisfying a user request is very difficult to determine because many different services and services combinations in the various libraries can fulfill the same requirements. We use an advanced Grid service trading to compute and find the best service or combination of services that answer the user requirements [8] taking into account the amount of calculations of given expression. As a benefit, users do not need to make explicit call to specific services over the Grid (GridRPC, Corba, etc..) and to know the exact name of the service they are looking for: they just describe the expression they want to compute in a given applicative domain.

The experiments of the interface have been carried out on the base of Armenian Grid [9] (SEE Grid site) and French National Grid'5000 infrastructures.

2 Introduction to the Grid-aware Web interface

The interface consists of the following three modules:

- A front end module which interacts with users
- The Grid service trading module that computes the service or the combination of services answering a user request. The execution of a service over the grid is nothing else then executing some BLAS, Lapack or Scalapack procedure or basic manipulation of numerical objects.
- The Solver module which performs computations and uses the DIET middleware for managing executions over the Grid

2.1 Front end module

The frontend module is the main module which interacts with users. Linear algebra encompasses methods and concepts that solve many different types of practical problems on the base of numerical objects (number, vector, and matrix). The module allows users to create numerical objects, as well as define and submit various linear algebra expressions. Both uploading and generation mechanisms have been developed for creation of numerical objects by specifying their types: scalars, vectors (ones, zeros, integer, etc...), matrices (triangular, identity, symmetric, etc.). The interface supports different operations (download, delete or change parameters) for already uploaded numerical objects. The names and the paths of the user-defined numerical objects are saved in the database. Only real and scalar types for numerical objects are implemented in the current experimental version.

2.2 Grid Service trading module

The purpose of the Grid trading module is to convert mathematical expressions into a list of procedures calls that answer the user request. In this experimental version, we only consider calling Scalapack procedures but other libraries can easily be added. Executions compute a service or a workflow of services equivalent to the initial mathematical expression. Today the use of LEX [10] and YACC [11] or FLEX and BISON [12] are widely spread among compiler developers. From specifications of the lexical and syntactical structure of a language, using regular expressions and LALR(1) grammars, these tools generate the corresponding analyses. A PHP program based on Bison/YACC has been developed, which checks and compares the parameters of the numerical objects from database and do syntactic and semantic analysis before converting the equation into a sequence of procedure calls.
Our service trading approach is based on a semantic description of services. It allows to compute the service or the combination of services that satisfy a user request. The input stream of the Grid service trading is a XML description that fully defines the given expression including the parameters and properties. The current module takes the expression from the front-end module, does syntactic and lexical analysis, converts the expression into the corresponding XML format, calls the Grid service trading service and as a result receives the list of Scalapack or BLAS functions that can solve the problem (note that the result may depend on the size of the object manipulated within the expression, e.g. calls to serial or parallel computational

services). Since that this approach is based on a semantic description of the libraries, it is easy to increase the number of libraries involved in our environment.

2.3 The Solver module

Several approaches exist for porting applications to grid platforms e.g. message-passing, batch processing, web portals, and Grid-RPC systems. In the case of the Grid-RPC model, clients submit computational requests to a scheduler or an agent that locates one or more servers available over the grid. The DIET Project aims at the development of a scalable middleware with initial efforts focused on distributing the scheduling problem across multiple agents. DIET consists into a set of elements that can be used together to build applications using the Grid-RPC paradigm. This middleware is able to find an appropriate server according to the information given in the client's request. The DIET environment consists of five different components: clients that submit problems to servers, servers that solve problems sent by clients, a database that contains information about software and hardware resources, a scheduler that chooses an appropriate server depending both of the problem sent and the information contained in the database, and finally monitors that get information about the status of the computational resources. The information stored on a server is the list of data it owns (eventually with their distribution and the way to access them), the list of problems that he can solve, and every information concerning its load (CPU capacity, available memory, etc.).
FORTRAN wrappers have been written for the Scalapack functions. After getting the list of functions to be called from the previous module, if the size of the data involved in the current call is not so large and can fits into the memory of the current machine, the expression is solved locally, otherwise and in most of cases, the Solver module, which acts as a DIET client, calls the corresponding Scalapack subprogram through a FORTRAN wrapper. The output of the current DIET call can be used as an input of the next procedure call. On Figure 1, we give the detailed structure of the interface.

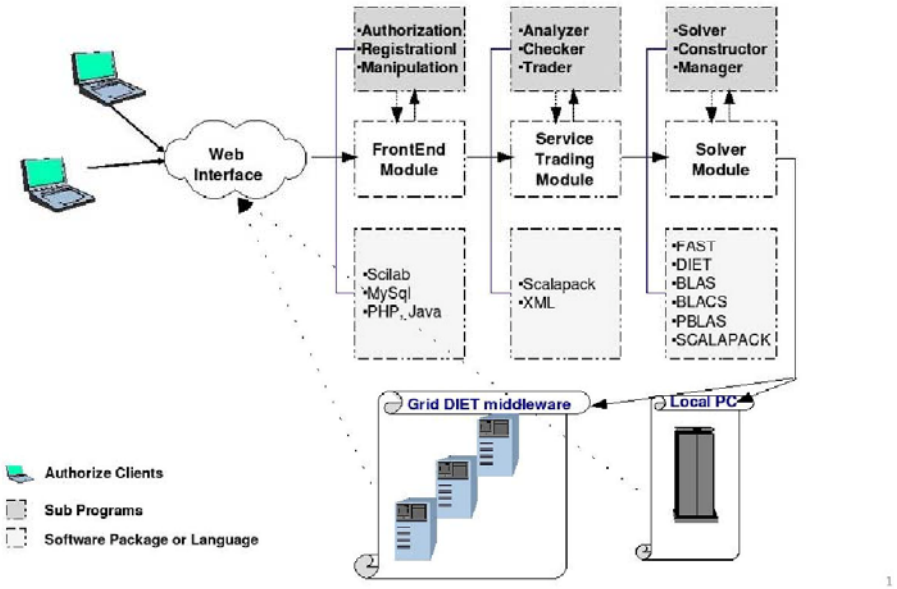

Fig 1: Interface structure

3 Example

As an illustration of the Interface, we consider a simple example. Assume that a user wants to solve the following algebraic expression C = A * (B * C), where A, B and C are general type 10000-by-10000 matrices. As mentioned above, the interface consists of the Front end, the Grid Service trading and the Solver modules. The interface then processes the request in the following way:

- After successful identification, the user uploads the numerical objects (A, B, C matrixes) into the Database through the uploading subprogram that includes a numerical object correctness checker. Via the "View numerical objects" subprogram, user can see and edit its own uploaded numerical objects. Only the names and the paths are saved in the database.
- The user enters the mathematical expression to be evaluated that involves the numerical objects initialized as: A * (B * C) and then submits it for execution.
- The syntactic and semantic analyzer subprogram based on Bison/YACC checks and compares the parameters (dimensions and names) of A, B, C matrices from the Database and does syntactic and semantic analyzing before converting the equation. If the expression is correct, the analyzer creates an XML file that describes the expression:
  <requete>

```
<domainName>LinearAlgebra</domainName>
<term>
 <expression>
  <operatorName>*</operatorName>
  <terms>
   <term>
    <variable>
     <name>A</name>
     <sorte>Matrix</sorte>
    </variable>
   </term>
   <term>
    <expression>
     <operatorName>*</operatorName>
     <terms>
       <term>
        <variable>
         <name>B</name>
         <sorte>Matrix</sorte>
        </variable>
       </term>
       <term>
        <variable>
         <name>C</name>
         <sorte>Matrix</sorte>
        </variable>
       </term>
      </terms>
    </expression>
   </term>
  </terms>
 </expression>
</term>
</requete>
```

- The service trading subprogram takes the XML file (fully defined expression) and converts it into the list of functions that can solve the requested expression. In our case the output is of the service trader is:

  PsGEMM ('N', 'N', 10000, 10000, 10000, 1., B, 1, 1, DESCB, C, 1, 1, DESCC, 0., p2, 1, 1, DESCp2) \\ p2= B * C
  PsGEMM ('N', 'N', 10000, 10000, 10000, 1 A, 1 1 DESCA, p2, 1, 1, DESCp2, 0., p1, 1, 1, DESCp1) \\ p1= A * p2
  p1

  Here the PsGEMM procedure performs one of the matrix-matrix operations: C= α*op(A)*op(B)+β*op(C), where α and β are scalars, op(A) and op(B) are

rectangular single precision real matrices of dimensions m-by-k and k-by-n respectively, C is a m-by-n matrix, and op(A) is a A or A^T.

- A list of FORTRAN wrapers have been developed for all PBLAS and BLAS functions. The FORTRAN programs initialize the process grid, distribute the matrix on the process grid, call correspondent PBLAS routine and then release the process grid. Input parameters of FORTRAN subprograms include the paths and files names of numerical objects. The output is stored saved into a standard temporary ASCII file that can be used as an input for the next FORTRAN subprogram. Taking into account the dimensions of the numerical object for given routine, the solver module defines where it is logical to execute the Scalapack subprogram and then executes it. If the dimensions of the current call are not too large and can fit into the memory of local PC, the expression is solved locally, otherwise and most of cases the Solver module performs executions over the grid using DIET.
- After successful executing all Scalapack subprograms, the last output is saved into the database. The final result is accessible from the interface and user can download it if necessary.

4 Conclusion

The full implementation of our environment for scientific computing over the Grid will allow users, who are not familiar with the parallel programming technologies and software tools (Grid & Cluster middlewares, MPI, JAVA, Unix OS, Open PBS, Condor, Scalapack, etc..), to create and submit their programs over a Web interface that hides all details of the underlying distributed infrastructure. Such an approach can be extended to any computational Grid that supports DIET (or another GridRPC type of middleware) and linear algebra libraries such as Scalapack. It can be easily extended to any application area where software libraries are available. Note that it is also a nice environment for demonstrating how computations can be transparently – from the user point of view – performed over the Grid.

5 Acknowledgment

The joint work has been done within the framework of INTAS YS Post Doctoral Fellowship (05-109-4390). It has also be supported the French Lego (ANR-05-CIGC-11) and ISTC A-1451 Projects.

References

1. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, An Updated Set of Basic Linear Algebra Subprograms, ACM Trans. Math. Soft., pp. 135-151, 28-2 2002
2. E.Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, Society for Industrial and Applied Mathematics, Third Edition, 0-89871-447-8 (paperback), 1999
3. L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, ScaLAPACK: A Linear Algebra Library for Message-Passing Computers, SIAM Conference on Parallel Processing, March 1997.
4. Bunks, J.P. Chancelier, F. Delebecque, C. Gomez, M. Goursat, R. Nikoukhah, S. Steer, Engineering and Scientific Computing with Scilab, Hardcover, ISBN: 978-0-8176-4009-5, pp. 491, 1999
5. Eddy Caron and Frédéric Desprez. DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. International Journal of High Performance Computing Applications, 20(3):335-352, 2006
6. J. Dongarra and R. Whaley, LAPACK Working Note 94: A User's Guide to the BLACS v1.0, University of Tennessee Computer Science Technical Report, UT-CS-95-281, March 1995, updated May 5, 1997 (Version 1.1)
7. Hrachya Astsatryan, Michel Daydé, Aurélie Hurault, Marc Pantel, Eddy Caron. On defining a Web Interface for Linear Algebra Tasks over Computational Grids. Dans / In: International Conference on Computer Science and Information Technologies (CSIT'07), Yerevan (Armenia), 24/09/2007-28/09/2007
8. Michel Daydé, Aurélie Hurault, Marc Pantel. Semantic-based Service trading: Application to Linear Algebra. Dans / In : VECPAR'06 - Workshop on Computational Grids and Clusters (WCGC 2006), Rio de Janeiro, Brazil, 10/07/2006-13/07/2006, Springer-Verlag, LNCS 4395, p. 622-633, 2006
9. H.V. Astsatryan, Yu. Shoukourian, V. Sahakyan, Creation of High-Performance Computation Cluster and DataBases in Armenia, Proceedings of the Second International Conference on Parallel Computations and Control Problems (PACO '2004), ISBN: 5-201-14974-X, pp. 466-471, Moscow, Russia, October 4-6, 2004
10. M. E. Lesk. Lex - a lexical analyzer generator, Computing Science Technical Report 39, AT&T Bell Laboratories, Murray Hill, NJ 1975
11. Steven C. Johnson. Yacc: yet another compiler compiler. CS Technical Report #32, Bell Telephone Laboratories, Murray Hill, NJ, 1975
12. Aaron Montgomery, Using Flex and Bison, http://www.mactech.com/articles/mactech/Vol.16/16.07/UsingFlexandBison/