GRID-ENABLING PROBLEM SOLVING ENVIRONMENTS: A CASE STUDY OF SCIRUN AND NETSOLVE

Michelle Miller * Christopher Moulding † Jack Dongarra ‡ Christopher Johnson §

Keywords: Grid Computing, Problem Solving Environments, Scientific Visualization, Numerical Libraries

Abstract

Combining the functionality of NetSolve, a grid-based middleware solution, with SCIRun, a graphically-based problem solving environment (PSE), yields a platform for creating and executing grid-enabled applications. Using this integrated system, hardware and/or software resources not previously accessible to a user become available completely behind the scenes. Neither the SCIRun system nor the SCIRun user need to know any details about how these resources are located and utilized. A SCIRun module merely makes an RPC-style call to NetSolve via the NetSolve C language API to invoke a certain routine and to pass its data. Distributed computation and the details of remote communication are completely abstracted away from the SCIRun framework and its end user.

1 OVERVIEW

The idea of a compute grid [5] is analogous to the idea of the electric power grid. One should be able to "plug in" to the compute grid to obtain computing services the way one can plug into the electric power grid to obtain electrical services. All the complicated details of electrical power generation, storage, and dissemination are hidden from the electricity user, who merely plugs devices requiring power into the electric power grid via a wall socket. In much the same way, the details of the compute grid need to be hidden from the computing user for ease of use. If applications cannot make use of computing infrastructure easily, the infrastructure will remain unused.

*Department of Computer Science, University of Tennessee, Knoxville, TN 37996 miller@cs.utk.edu

†School of Computing, University of Utah, Salt Lake City, UT 84112 moulding@cs.utah.edu

‡Department of Computer Science, University of Tennessee, Knoxville, TN 37996 dongarra@cs.utk.edu

§School of Computing, University of Utah, Salt Lake City, UT 84112 crj@cs.utah.edu

However, there are many issues involved in grid-enabling pre-existing software. Grid computing seeks to go beyond mere networked computing by providing some infrastructure to support the use of resources in the way one would access/utilize a single machine, instead of a network of them. For example, Globus [4] provides a number of services which resemble the services provided by a traditional single machine operating system, such as UNIX. The question is how to build applications or executables for such a system. Usually, one compiles executables on the target architecture (*e.g.,* Linux/x86, Linux/Alpha), so that the executable is properly configured to run on that platform (or even on a specific machine). To extend the operating system metaphor to the compute grid, we must ask how a grid application is constructed, compiled, or executed. Must it be written from scratch to take into account that its runtime environment will be a grid or can an existing application be easily modified to be grid-enabled? This question of writing grid-enabled applications really extends our notion of portable code. Taking into consideration running in a grid environment, truly portable code must be able to run on many hardware platforms as well as a variety of metacomputing environments, such as Condor [8], Globus, Legion [6], Ninf [11], and NetSolve. Is this a reasonable burden to place on application code? How can we make grid environments easier to program to?

NetSolve [2], with its simple RPC semantics, exemplifies an easy-to-use **plug** to enable applications to utilize the compute grid. NetSolve is a grid-enabled middleware system that allows users to access computational resources, both hardware and software, that are distributed across a network. It provides mechanisms to permit users to call remote libraries simply. Scientists and engineers can make use of libraries without the need to locate, configure, compile, install, and upgrade these libraries.

SCIRun [10] is a Problem Solving Environment (PSE) that could benefit from having access to the compute grid, particularly access to remote numerical libraries. SCIRun is a scientific problem solving environment that allows the interactive construction, debugging, and steering of large-scale, often parallel, scientific computations. SCIRun can be envisioned as a "computational workbench," in which a scientist can design and modify simulations interactively

via a component-based visual programming tool.

We provide a case study of grid-enabling a problem solving environment, in this case using NetSolve to grid-enable SCIRun. Using NetSolve as a mechanism to plug SCIRun into the grid is simple and fairly unobtrusive to the PSE. SCIRun can leverage grid software resources, such as existing tuned numerical libraries, without needing to fold library source code into its releases. Also, SCIRun does not need to worry about portability for these libraries. NetSolve facilitates heterogeneous environments; a user can run a client from any supported architecture regardless of the platform running the server (library). In this way, NetSolve solves the portability problem: libraries that are tied to one or two machine architectures can be used by a NetSolve client running on almost any type of machine.

2 INTRODUCTION TO SCIRUN AND NETSOLVE

2.1 SCIRun

SCIRun Philosophy

SCIRun is designed to provide high-level control over parameters in an efficient and intuitive way, through graphical user interfaces and scientific visualization. The cause-effect relationships within the simulation become more evident as the scientist adjusts parameters, thus allowing the scientist to develop more intuition about the effect of problem parameters, to detect program errors, to develop insight into the operation of an algorithm, or to deepen an understanding of the physics of the problem(s) being studied. SCIRun fosters the asking of "What if?" questions.

SCIRun attempts to be a single-stop, usable, interactive problem solving environment for scientists and engineers. SCIRun provides a "whole" solution by integrating the normally discrete processing phases of modeling, simulation, and visualization into one problem solving environment. An integrated system frees the user from learning multiple systems and the data conversion headaches of moving data from one program to another. Next, in presenting a visual programming language, SCIRun appeals to the engineer/scientist with little "traditional" programming knowledge. Finally, SCIRun allows a user to interact with the running simulation by manipulating various parameters, in effect guiding or steering the computation. These mechanisms allow a user to adjust the computation without losing all of the calculations already performed, but only those that need to be redone due to changes in the parameter space. We believe that allowing a scientist more interaction with her simulation yields a more satisfying scientific tool.

SCIRun Architecture

The SCIRun architecture is built around the metaphor of a dataflow graph of computational modules constructed by the user, which is in effect a visual program. A sample application program written using the PSE is shown in Figure 1. The modules are at a coarse grain of functionality. The modules, represented visually as boxes, are linked together using wires (*data pipes*). Data flows from modules at the top of the graph to modules at the bottom of the graph. The linkage enforces data type checking, as one cannot wire together a module with an output parameter that does not match the input parameter of the downstream module.

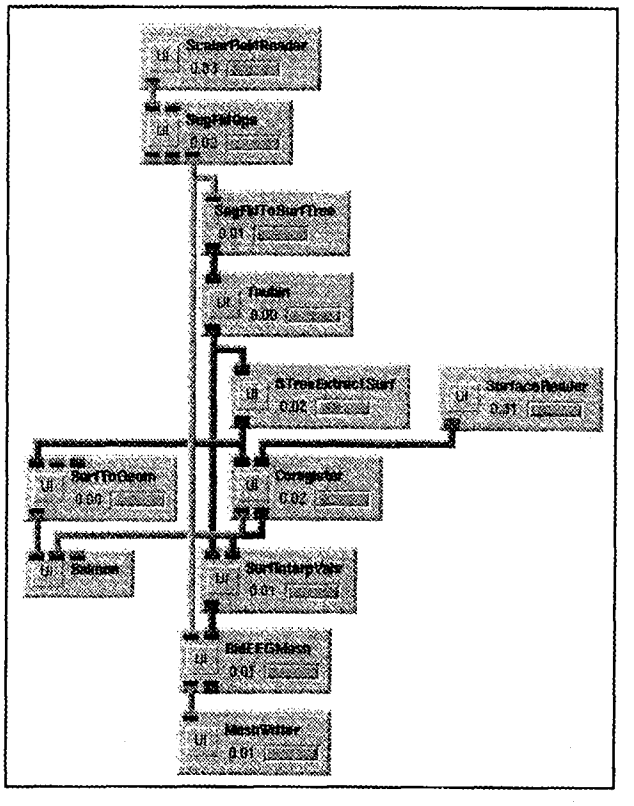

Figure 1: An example SCIRun network of a large-scale biomedical simulation shows the dataflow programming interface.

Execution is multi-threaded to optimize performance. Each SCIRun module has its own thread of execution and runs as soon as it receives all of its input parameters. In other words, concurrent execution based on data dependency analysis, or task parallelism, is provided. SCIRun is targeted toward shared memory symmetric multiprocessor (SMP) machines such as the SGI Origin 2000 and SMP Linux PCs. SCIRun limits itself to the shared memory environment to avoid using copying and/or marshaling to pass data between the various components that make up the SCIRun system. This self-imposed limit

makes it difficult to extend SCIRun to be usable in a distributed memory environment.

Computational steering is achieved by a user either manipulating a geometrical object, or widget, within the visualization window or by altering a parameter associated with a module through that module's GUI. A dataflow dependency analysis is then performed to identify the extent of the change to the dataflow graph. Only the modules affected by the change will be recomputed. In this way, a user can interact with her simulation without losing work unnecessarily.

2.2 NetSolve

NetSolve Philosophy

NetSolve is designed to be a simple-to-use middleware system. NetSolve allows users to access additional hardware and/or software resources available remotely. In this way, it promotes the sharing of these resources within and between research communities in the computational sciences. A scientist can make calls to a library routine without worrying about where the code and execution resources reside. Through the use of an intelligent agent, NetSolve automatically selects the most appropriate machine to service the client request (based on criteria such as service availability, network locality, and machine workload).

A simple RPC-style call gives access to complicated software routines. NetSolve tracks which machines have computational servers running and with which library services they are provisioned. NetSolve also tracks the workload of each server to yield the best choice of server for a given client request. In other words, NetSolve takes care of the details of finding a machine on which to run the software routine. In addition, NetSolve provides the ability to harness diverse machine architectures to work together on a computation (*heterogeneous computing*).

NetSolve seeks to reduce the headaches of maintaining software libraries. It addresses portability by permitting non-portable code to be run from virtually any machine architecture via a client call. In addition, legacy codes can be easily wrapped for use with new systems by using NetSolve.

NetSolve provides a "blue collar grid computing" solution by tying together PSEs and grid resources (metacomputing environments). As shown in Figure 2, NetSolve acts as a middleware plug to the grid by supporting interfaces to various front-ends, such as PSEs, and to various grid back-ends, such as Globus or NetSolve itself.

NetSolve's interfaces to PSEs such as Matlab and Mathematica extend the functionality of PSEs to software and hardware not supported by the native PSEs. In this way, NetSolve enables PSEs to make use of software without requiring that the routines be provided by the

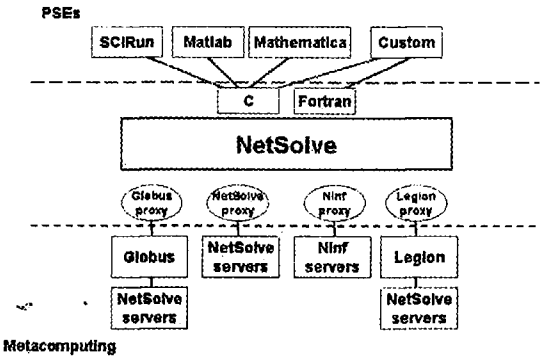

Figure 2: NetSolve as the middleware connecting PSEs to grid environments.

PSE. For example, a user could invoke ScaLAPACK routines on an Massively Parallel Processor (MPP) through a NetSolve call within a Matlab interface even though Matlab is not natively MPP-enabled. In this way, we can extend the hardware and software on which these PSEs can operate (*i.e.*, run routines on platforms outside of those supported by the PSE).

NetSolve Architecture

The NetSolve architecture is based on a client/server design with an intelligent agent that keeps the state of the system so a client only needs to know where an agent is located. NetSolve works through the use of persistent servers that service requests made by a user call invoking the NetSolve client. Each NetSolve server is pre-configured to run a set of services (or wrapped library routines) at compile time. The server registers itself and its set of services with the NetSolve agent with which it chooses to be associated upon startup. Then, the agent takes in client service requests and returns the "best" server for the client to use based on proximity and machine workload. Next, the client sends the library routine call and data to the server for processing. Finally, the server returns the result to the client.

NetSolve provides simple RPC-style calls as its client API. All a client sees is that it makes a call and a result is returned. All the intelligence in the system happens "behind the scenes." NetSolve supports APIs for C and Fortran programming languages, as well as for Matlab and Mathematica problem solving environments to facilitate use by programmers and non-programmers alike.

In addition, NetSolve provides extensible service creation. NetSolve provides a Problem Description File (PDF), which is merely an Interface Definition Language (IDL), as the mechanism for generating wrappers to library code. The various problems described in a PDF then

become NetSolve services which can be enabled by server instances. In this way, one can create NetSolve services that can be run by NetSolve servers from any stand-alone code module.

Different proxies support multiple metacomputing back-ends. NetSolve supports different grid computing back-ends through the use of client proxies which can interface to these various back-ends. For now, one cannot mix back-ends, but must choose one grid back end when the client is installed.

All elements of NetSolve run on most UNIX platforms. In addition, NetSolve has a Windows 95/98/NT client interface.

3 GRID-ENABLING THE SCIRUN PSE

Must a grid application be written from scratch to take into account that its runtime environment will be a grid or can an existing application be easily modified to be grid-enabled? We believe it is not possible in practice to write an application completely independent of its target execution environment. There are too many assumptions that must be made about the environment to make a totally general piece of code. However, grid environments should not be complex to program. NetSolve provides a middle layer between an applications programmer/PSE and a grid environment to abstract away the details of interacting with a grid. Therefore, we believe it must present the simplest of interfaces to programmers/users.

NetSolve, as grid middleware, provides an easy-to-use plug for an existing application to grid-enable itself. However, grid-enablement is not automatic — application programmers must augment their code with calls to use grid resources.

3.1 Methods

Our goal is to allow a SCIRun user to easily choose whether or not to use NetSolve within their application. Creating a set of grid-runnable components (modules) that are either interchangeable with pre-existing SCIRun components or are new components is a clear, but unobtrusive way of integrating NetSolve capabilities into the SCIRun PSE. A SCIRun applications programmer must target their application to be run on the grid by selecting grid-enabled modules. Thus the complexity of using the grid is reduced to the much more simple problem of calling a NetSolve-enabled module. As far as the task of writing grid-enabled SCIRun modules, existing modules can be used as templates for the NetSolve-enabled modules, which merely require the addition of a few calls to NetSolve.

Our initial approach is to extend the functionality of a few SCIRun modules in a domain-specific way. Initially, we built a NetSolve-enabled sparse matrix solver module; work has already begun to extend this approach to dense problems as well. The NetSolve-enabled matrix solver modules use highly tuned numerical libraries which are accessible through NetSolve's client API. In this way, we extend the functionality of the SCIRun framework, while also providing a mechanism for distributing the compute-intensive code (in this case the solve) to an MPP. Using NetSolve to call these sparse and dense numerical library routines allows the NetSolve user the ability to run MPI [12] on distributed memory processors without having to learn the interstices of a message passing numerical library (*i.e.*, the SCIRun developers need not find a way to merge SCIRun's multi-threaded environment with MPI). Since each module in SCIRun has a separate thread of execution, the module functions essentially atomically as a single unit of computation. This execution model fits well with NetSolve's black box approach. NetSolve takes care of finding a server that can run the package (*e.g.*, PETSc [7]), as well as handling the details of MPP execution (*e.g.*, starting the distributed memory run, passing the data back and forth, *etc.*), so that these details are abstracted away from the PSE.

The NetSolve-enabled modules are patterned after the existing SCIRun matrix solver module, SolveMatrix, that supports sparse matrices and provides several iterative methods (*e.g.*, variations of conjugate gradient, Jacobi, Richardson, and minimal residual methods). The original module allows a user to choose the method to be used and to adjust parameters such as error tolerance, maximum number of iterations, and preconditioner. The NetSolve SparseSolve module allows the user to choose a numerical package to use for the solve as well setting the error tolerance and maximum number of iterations. Sparse iterative and sparse direct methods are supported, enabling a user to choose either PETSc or Aztec for an iterative solve or MA28 or SuperLU for a direct solve. Depending on the method chosen, a netsolve call is formatted and a client request created for one of the services listed above. We used a NetSolve blocking call, so the call waits until it receives the result back from whichever server the NetSolve agent assigns to the request. The user must specify a NetSolve agent in her environment before executing this module. Also, we had to build and link the NetSolve client library into the SCIRun PSE executable.

In general, a module writer must translate between the SCIRun object/datatype (C++ class) and the parameters required by the NetSolve API to the target numerical library or user-defined service. A SCIRun module receives its input from input ports of particular types and sends out its output through output ports that are also typed. So, input data must be converted for the call to NetSolve, and

the output from NetSolve must be converted back into the SCIRun datatype.

In our specific case, we sought to make components plug-compatible or interchangeable which required us to preserve the SCIRun solver "interface" (*i.e.*, the number and types of module inputs and outputs). Both modules take essentially the same input: a sparse matrix and a vector (right-hand side), although it is conceivable that one could wire a dense matrix into the normal SCIRun solver, that is explicitly prohibited with the NetSolve SparseSolve module. A dense matrix would be treated as a sparse matrix by the normal SCIRun solver, so it is not pragmatic to do so. Both modules output a vector (left-hand side). So, the original SCIRun module and the NetSolve-enable SparseSolve module are interchangeable within any given SCIRun application.

3.2 Preliminary Results

Creating a SCIRun module to make use of a NetSolve service was simple. Three lines of code enabled a SCIRun module to utilize NetSolve for the numerical solve. We created a SparseSolve module which can be used in place of the normal SCIRun SolveMatrix module. The only difference is that the module uses NetSolve to call a standard sparse package to do the solve. Preliminary results reveal cases where the NetSolve-enabled PETSc service allows SCIRun users to solve problems that could not be solved using the unaugmented PSE. These findings warrant further investigation.

4 RELATED RESEARCH

Work investigating grid-enabling entire problem solving environments is relatively new, with many open research questions. The most popular and widespread Problem Solving Environments (PSEs) are "programmed" in the language of math: Matlab, Mathematica, and Maple. NetSolve provides interfaces to Matlab and Mathematica, allowing users of these PSEs to leverage the power of the grid via NetSolve (*i.e.*, gain access to additional hardware and software that is not provided within these systems).

As far as work investigating running applications on the grid, most of the work has focused on writing new applications which presuppose the existence of a grid. This allows them to program to the specifics of a particular grid environment. The Globus team wrote an interesting grid application that gathers data from a remote high-end instrument, a synchrotron light source; configures the resources needed to perform tomographic image reconstruction from grid resources; and allows a user to steer the computation by selecting and altering parameter values and to visualize the results [13].

However, some investigations have used some pre-existing code to cobble together grid applications. Casanova, *et al.* [3] grid-enabled an existing Monte Carlo simulator for neuron functions called MCell. They used AppLeS [1] and NetSolve to collect resources and schedule a massively parallel parameter space search that allowed biomedical researchers to make strides in their research.

Finally, previous work on a distributed version of the SCIRun PSE [9] investigated executing portions of a SCIRun application program remotely. This work showed feasibility, but the work stopped short of grid-enabling the SCIRun problem solving environment. However, since it was woven into the fabric of the SCIRun runtime environment, this work provided a more generalized solution to distributed computing within SCIRun.

5 CONCLUSIONS

To answer the question about burdens on application programmers to achieve portability, we must conclude that indeed some design must be targeted toward grid environments, but grid environments need to be easy to plug into. In our case, we created separate modules to achieve our grid-enablement of SCIRun. But we found that it is simple and feasible to create SCIRun modules that use NetSolve to utilize remote services. With a few simple changes, a SCIRun user gains access to grid software services, in this case numerical libraries.

A PSE and its users gain access to software packages without having to support these libraries. In this way, the functionality of a PSE is greatly increased without the usual increase in maintenance and packaging work usually needed to fold library code into the software to provide a turnkey system. For example, the SCIRun source tree does not need to keep or distribute the PETSc source in order to make use of the set of PETSc functionality accessed by NetSolve. Lastly, with the NetSolve interface in place, it will now be relatively easy to access other grid platforms, such as Condor, Globus, Legion, and Ninf, through the NetSolve plug since NetSolve is building interfaces to these metacomputing back-ends.

6 INTEGRATION ISSUES AND FUTURE WORK

It would be nice to have a user select modules to be run remotely and have the proper code emitted to configure a NetSolve server comprised of the modules selected, and have the NetSolve client call formatted and invoked automatically (under the hood). Unfortunately, the SCIRun infrastructure gets in our way. There is just no easy way

to hook NetSolve into the persistent I/O SCIRun routines. It is a simple to add the sockets calls at this level, but we need a layer of abstraction above raw sockets or file manipulations to automatically emit NetSolve code that could execute normal SMP-targeted SCIRun code remotely.

In addition, a number of areas of future work in NetSolve would help the integration with SCIRun. First, NetSolve does not support steering. NetSolve services are a black box. There is no way within NetSolve to halt a computation while it is running and restart it with new parameters, in other words to *steer* the computation. Guiding the computation in this way is one of the hallmarks of SCIRun, so allowing user interaction is important. Second, a variety of interfaces from NetSolve to numerical libraries should be supported. SCIRun needs an interface that exposes lots of parameters which a user can interact with or steer; whereas, a typical Matlab user of NetSolve may want the simplest interface to a library. One can think of these as expert and non-expert APIs.

Acknowledgments: The authors would like to thank Terry Moore and Clint Whaley for their helpful comments, as well as Dave Weinstein for the use of his figure of a SCIRun application. SCIRun receives funding from the NSF, DOE, and NIH. NetSolve receives funding from the NSF under GRANT #ACI-9876895.

References

[1] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing '96*, 1996.

[2] H. Casanova and J. Dongarra. NetSolve: A network server for solving computational science problems. *International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997. Also in Proceedings of Supercomputing '96, Pittsburgh.

[3] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proceedings of SuperComputing 2000 (SC'00)*, Nov. 2000. to appear.

[4] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[5] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, 1999.

[6] A.S. Grimshaw, W.A. Wulf, and the Legion Team. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, 1997.

[7] W.D. Gropp and B.F. Smith. The design of data-structure-neutral libraries for the iterative solution of sparse linear systems. *Scientific Programming*, 5:329–336, 1996.

[8] M. Litzkow, M. Livny, and M.W. Mutka. Condor – a hunter of idle workstations. In *Proceedings of the Eighth International Conference of Distributed Computing Systems*, pages 104–111, 1988.

[9] M. Miller, C. Hansen, and C.R. Johnson. Simulation steering with SCIRun in a distributed environment. In B. Kgstrm, J. Dongarra, E. Elmroth, and J. Wasniewski, editors, *Applied Parallel Computing, 4th International Workshop, PARA'98*, volume 1541, pages 366–376. Springer-Verlag, Berlin, 1998.

[10] S.G. Parker and C.R. Johnson. SCIRun: A scientific programming environment for computational steering. In *Proceedings of Supercomputing '95*. IEEE Computer Society Press, 1995.

[11] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima. Ninf: Network based information library for globally high performance computing. In *Proceedings of Parallel Object-Oriented Methods and Applications*, pages 39–48. Santa Fe, Feb. 1996.

[12] M Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*, volume 1, The MPI-1 Core. MIT Press, second edition, 1998.

[13] G. von Laszewski, J.A. Insley, I. Foster, J. Bresnahan, C. Kesselman, M. Su, M. Thiebaux, M.L. Rivers, S. Wang, B. Tieman, and I. McNulty. Real-time analysis, visualization, and steering of microtomography experiments at photon sources. In *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, Apr. 1999.