Providing Access to High Performance Computing Technologies

Jack Dongarra[1], Shirley Browne[2], and Henri Casanova[2]

[1] University of Tennessee and Oak Ridge National Laboratory
[2] University of Tennessee, Knoxville TN 37996 USA

Abstract. This paper describes two projects underway to provide users with access to high performance computing technologies. One effort, the National HPCC Software Exchange, is providing a single point of entry to a distributed collection of domain-specific repositories. These repositories collect, catalog, evaluate, and provide access to software in their specialized domains. The NHSE infrastructure allows these repositories to interoperate with each other and with the top-level NHSE interface. Another effort is the NetSolve project which is a client-server application designed to solve computational science problems over a network. Users may access NetSolve computational servers through C, Fortran, MATLAB, or World Wide Web interfaces. An interesting intersection between the two projects would be the use of the NetSolve system by a domain-specific repository to provide access to software without the need for users to download and install the software on their own systems.

1 The National HPCC Software Exchange

1.1 Overview of the NHSE

The National HPCC Software Exchange (NHSE) is an Internet-accessible resource that facilitates the exchange of software and of information among research and computational scientists involved with High Performance Computing and Communications (HPCC) [1] [3]. The NHSE facilitates the development of discipline-oriented software repositories and promotes contributions to and use of such repositories by Grand Challenge teams, as well as other members of the high performance computing community.

The expected benefits from successful deployment of the NHSE include the following:

- Faster development of better-quality software so that scientists can spend less time writing and debugging programs and more time on research problems.
- Reduction of duplication of software development effort by sharing of software.
- Reduction of time and effort spent in locating relevant software and information through the use of appropriate indexing and search mechanisms and domain-specific expert help systems.

[3] `http://www.netlib.org/nhse/`

- Reduction of duplication of effort in evaluating software by sharing software review and evaluation information.

The scope of the NHSE is software and software-related artifacts produced by and for the HPCC Program. Software-related artifacts include algorithms, specifications, designs, and software documentation. The following three types of software being made available:

- Systems software and software tools. This category includes parallel processing tools such as parallel compilers, message-passing communication subsystems, and parallel monitors and debuggers.
- Basic building blocks for accomplishing common computational and communication tasks. These building blocks will be of high quality and transportable across platforms. Building blocks are meant to be used by Grand Challenge teams and other researchers in implementing programs to solve computational problems. Use of high-quality transportable components will speed implementation, as well as increase the reliability of computed results.
- Research codes that have been developed to solve difficult computational problems. Many of these codes will have been developed to solve specific problems and thus will not be reusable as is. Rather, they will serve as proofs of concept and as models for developing general-purpose reusable software for solving broader classes of problems.

1.2 Domain-specific Repositories

The effectiveness of the NHSE will depend on discipline-oriented groups and Grand Challenge teams having ownership of domain-specific software repositories. The information and software residing in these repositories will be best maintained and kept up-to-date by the individual disciplines, rather than by centralized administration. Domain experts are the best qualified to evaluate, catalog, and organize software resources within their domain.

Netlib – Mathematical Software An example of a domain-specific repository is the Netlib mathematical software repository, which has been in existence since 1985 [2]. Netlib differs from other publicly available software distribution systems, such as Archie, in that the collection is moderated by an editorial board and the software contained in it is widely recognized to be of high quality. Netlib distributes freely-available numerical libraries such as EISPACK, LINPACK, FFTPACK, and LAPACK that have long been used as important tools in scientific computation. The Netlib collection also includes a large number of newer, less well-established codes. Software is available in all the major numerical analysis areas, including linear algebra, nonlinear equations, optimization, approximation, and differential equations. Most of the software is written in Fortran, but programs in other languages, such as C and C++, are also available. Netlib uses the Guide to Available Mathematical Software (GAMS) classification system [3] to help users quickly locate software that meets their needs.

A branch of Netlib specialized to high performance computing, called HPC-netlib, is currently under development. HPC-netlib will provide access to algorithms and software for both shared memory and distributed memory machines, as well as to information about performance of parallel numerical software on different architectures.

PTLIB – Parallel Tools Another domain-specific repository that is under development is the PTLIB parallel tools repository. PTLIB will provide access to high-quality tools in the following areas: communication libraries, data parallel language compilers, automatic parallelization tools, debuggers and performance analyzers, parallel I/O, job scheduling and resource management.

1.3 Repository Interoperation

In addition to providing access to its own software, a repository may wish to import software descriptions from other repositories and make this software available from its own interface. For example, a computational chemistry repository may wish to provide access to mathematical software and to parallel processing tools in a manner tuned to the computational chemistry discipline. A repository interoperation architecture is shown in Figure 1.

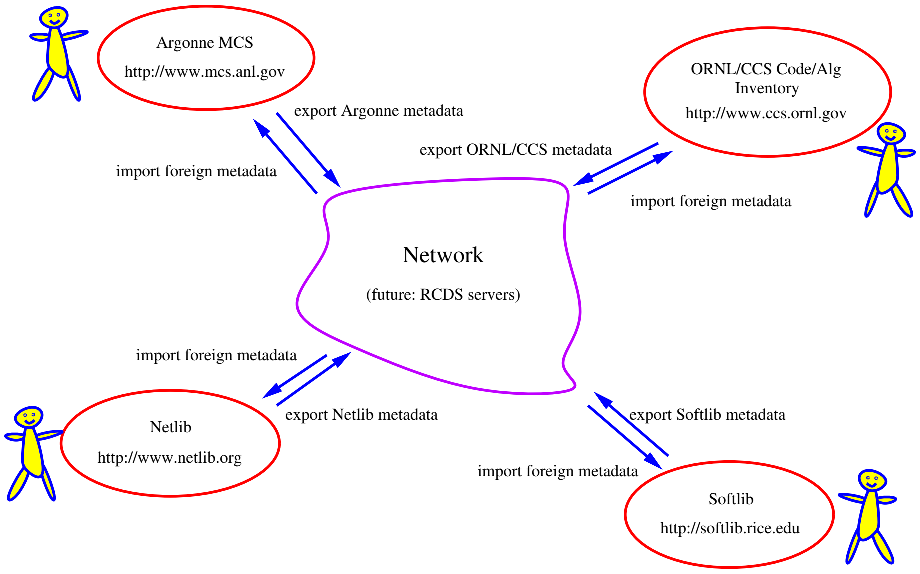

Fig. 1. Repository Interoperation Architecture

The NHSE is using the Reuse Library Interoperability Group's Basic Interoperability Data Model (BIDM) as its interoperability mechanism [4]. Participating HPCC repositories and some individual contributors have placed META and LINK tags in the headers of HTML files that describe their software resources. This information may then be picked up by other repositories and incorporated into their own software catalogs. The NHSE is developing a toolkit called Repository in a Box (RIB) that will assist repository maintainers in creating and maintaining software catalog records, in exchanging these records with other repositories (including the top-level virtual NHSE repository), and in providing a user interface to their software catalog. The Resource Cataloging and Distribution System under development at the University of Tennessee will provide a scalable substrate for repository interoperability by providing catalog and location servers that map resource names to catalog and location information.

1.4 Software Review Framework

The NHSE has designed a software review policy that enables easy access by users to information about software quality, but which is flexible enough to be used across and specialized to different disciplines. The three review levels recognized by the NHSE are the following: Unreviewed, Partially reviewed, and Reviewed. The *Unreviewed* designation means only that the software has been accepted into the owning repository and is thus within the scope of HPCC and of the discipline of that repository. The *Partially reviewed* designation means that the software has been checked by a librarian for conformance with the scope, completeness, adequate documentation, and construction guidelines. The *Reviewed* designation means that the software has been reviewed by an expert in the appropriate field, for example by an author of a review article in the electronic journal *NHSE Review* [4], and found to be of high quality. Domain-specific repositories and expert reviewers are expected to refine the NHSE software review policy by adding additional review criteria, evaluation properties, and evaluation methods and tools. The NHSE also provides for soliciting and publishing author claims and user comments about software quality. All software exported to the NHSE by its owning repository or by an individual contributor is to be tagged with its current review level and with a pointer to a review abstract which describes the software's current review status and includes pointers to supporting material.

2 The NetSolve project

An ongoing thread of research in scientific computing is the efficient solution of large problems. Various mechanisms have been developed to perform computations across diverse platforms. The most common mechanism involves software libraries. Unfortunately, the use of such libraries presents several difficulties. Some software libraries are highly optimized for only certain platforms and do not provide a convenient interface to other computer systems. Other libraries

[4] `http://nhse.cs.rice.edu/NHSEreview/`

demand considerable programming effort from the user, who may not have the time to learn the required programming techniques. While a limited number of tools have been developed to alleviate these difficulties, such tools themselves are usually available only on a limited number of computer systems. MATLAB [5] is an example of such a tool.

These considerations motivated the establishment of the NetSolve project. NetSolve is a client-server application designed to solve computational science problems over a network. A number of different interfaces have been developed to the NetSolve software so that users of C, Fortran, MATLAB, or the World Wide Web can easily use the NetSolve system. The underlying computational software can be any scientific package, thus helping to ensure good performance through choice of an appropriate package.. Moreover, NetSolve uses a load-balancing strategy to improve the use of the computational resources available. Some other systems are currently being developed to achieve somewhat similar goals. Among them are the Network based Information Library for high performance computing (Ninf) [6] project which is very comparable to NetSolve in its way of operation, and the Remote Computation System (RCS) [7] which is a remote procedure call facility for providing uniform access to a variety of supercomputers.

We introduce the NetSolve system, its architecture and the concepts on which it is based. We then describe how NetSolve can be used to solve complex scientific problems.

2.1 The NetSolve System

The NetSolve system is a set of loosely connected machines. By *loosely* connected, we mean that these machines can be on the same local network or on an international network. Moreover, the NetSolve system can be running in a *heterogeneous* environment, which means that machines with different data formats can be in the system at the same time.

The current implementation sees the system as a completely connected graph without any hierarchical structure. This initial implementation was adopted for simplicity and is viable for now. Our current idea of the *NetSolve world* is of a set of independent NetSolve systems in different locations, possibly providing different services. A user can then contact the system he wishes, depending on the task he wants to have performed and on his own location. In order to manage efficiently a pool of hosts scattered on a large-scale network, future implementations might provide greater structure (e.g., a tree structure), which will limit and group large-range communications.

Figure 2 shows the global conceptual picture of the NetSolve system. In this figure, a NetSolve client send a request to the NetSolve agent. The agent chooses the "best" NetSolve resource according to the size and nature of the problem to be solved.

Several instances of the NetSolve agent can exist on the network. A good strategy is to have an instance of the agent on each local network where there are NetSolve clients. Of course, this is not mandatory; indeed, one may have only a single instance of the agent per NetSolve system.

Every host in the NetSolve system runs a NetSolve *computational* server (also called a *resource*, as shown in Figure 2). The NetSolve resources have access to scientific packages (libraries or stand-alone systems).

An important aspect of this server-based system is that each instance of the agent has its own *view* of the system. Therefore, some instances may be aware of more details than others, depending on their locations. But eventually, the system reaches a stable state in which every instance possesses all the available information on the system.

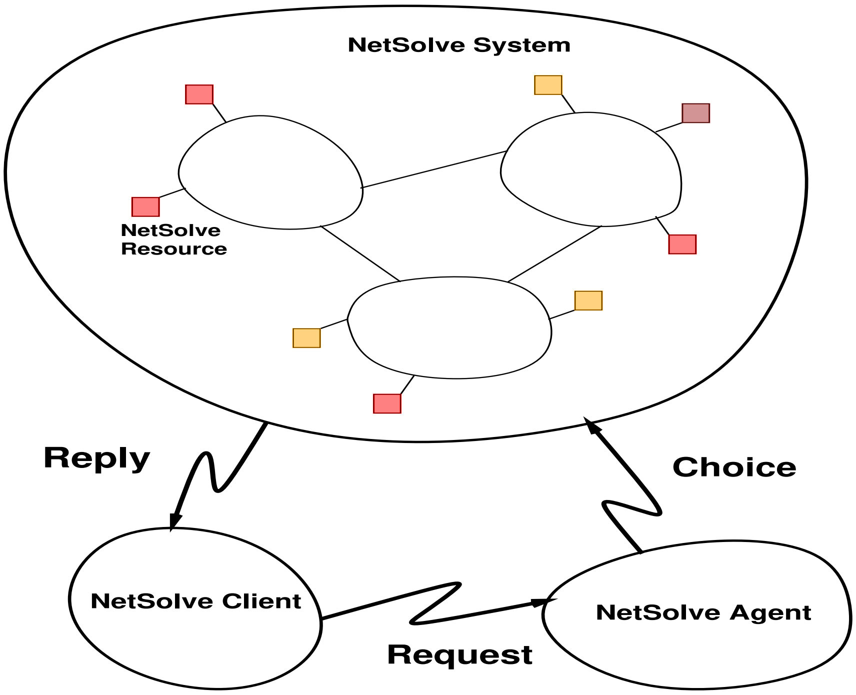

Fig. 2. The NetSolve System

Communication within the NetSolve system is achieved through the TCP/IP socket layer and heterogeneous environments are supported thanks to the XDR protocol [8].

2.2 Problem Specification

To keep NetSolve as general as possible, we needed to find a formal way of describing a problem. Such a description must be carefully chosen, since it will affect the ability to interface NetSolve with arbitrary software.

A problem is defined as a 3-tuple: $< name, inputs, outputs >$, where

- $name$ is a character string containing the name of the problem
- $inputs$ is a list of input objects
- $outputs$ is a list of output objects

An object is itself described as follows: $< object, data >$, where $object$ can be 'MATRIX', 'VECTOR' or 'SCALAR' and $data$ can be any of the standard FORTRAN data types. This description has proved to be sufficient to interface NetSolve with numerous software packages. The NetSolve administrator can then not only choose the best platform on which to install NetSolve, but also select the best packages available on the chosen platform.

The current installation of NetSolve at the University of Tennessee uses the BLAS [9], [10], [11], LAPACK [12], ItPack [13], LINPACK [14] and FitPack [15]. These packages are available on a large number of platforms and are freely distributed. The use of ScaLAPACK [16] on massively parallel processors would be a way to use the power of high-performance parallel machines via NetSolve.

2.3 Client Interfaces

One of the main goals of NetSolve is to provide the user with a large number of interfaces and to keep them as simple as possible.

The MATLAB Interface :

We developed a MATLAB interface which provides interactive access to the NetSolve system. Interactive interfaces offer several advantages. First, they are easy to use because they completely free the user from any code writing. Second, the user still can exploit the power of software libraries. Third, they provide good performance by capitalizing on standard tools such as MATLAB. Let us assume, for instance, that MATLAB is installed on one machine on the local network. It is possible to use NetSolve via the MATLAB interface on this machine and in fact use the computational power of another more powerful machine where MATLAB is not available.

Within MATLAB, NetSolve may be used in two ways. It is possible to call NetSolve in a *blocking* or *nonblocking* fashion. Here is an example of the MATLAB interface to solve an linear system computation using the blocking call:

```
>> a = rand(100); b = rand(100,1);
>> x = netsolve('ax=b',a,b)
```

This MATLAB script first creates a random 100×100 matrix, a, and a vector b of length 100. The call to the `netsolve` function returns with the solution. This call manages all the NetSolve protocol, and the computation may be executed on a remote host.

Here is the same computation performed in a nonblocking fashion:

```
>> a = rand(100); b = rand(100,1);
>> request = netsolve_nb('send','ax=b',a,b)
>> x = netsolve_nb('probe',request)
     Not Ready Yet
>> x = netsolve_nb('wait',request)
```

Here, the first call to `netsolve_nb()` sends a request to the NetSolve agent and returns immediately with a request identifier. One can then either *probe* for the request or *wait* for it. Probing always returns immediately, either signaling that the result is not available yet or, if available, stores the result in the user data space. Waiting blocks until the result is available and then store it in the user data space. This approach allows user-level parallelism and communication/computation overlapping (see Section 2.4).

Other functions are provided, for example, to obtain information on the problems available or on the status of pending requests.

C and FORTRAN interfaces :

In addition to the MATLAB interface, we have developed two programming interfaces, one for Fortran and one for C. Unlike the interactive interfaces, programming interfaces require some programming effort from the user. But again, with a view to simplicity, the NetSolve libraries contain only a few routines, and their use has been made as straightforward as possible. As in MATLAB, the user can call NetSolve *asynchronously.*

A very attractive feature of these interfaces is that NetSolve preserves the original calling sequence of the underlying numerical software. It is then almost immediate to convert a code to NetSolve, as shown in the short FORTRAN example below :

```
C     Linear system solve : Call to LAPACK

      call DGESV(N,1,A,MAX,IPIV,B,MAX,INFO)

C     Linear system solve : Call to NetSolve

      call FNSOLVE('DGESV',NSINFO,
                   N,1,A,MAX,IPIV,B,MAX,INFO)
```

2.4 Performance

One of the challenges in designing NetSolve was to combine ease of use and excellence of performance. Several factors ensure good performance without increasing the amount of work required of the user. In addition to the availability of diverse scientific packages (as discussed in a preceding section), these factors include load balancing and the use of simultaneous resources.

Load Balancing :

Load balancing is one of the most attractive features of the NetSolve project. NetSolve performs computations over a network containing a large number of machines with different characteristics, and one of these machines is the most suitable for a given problem, meaning the one yielding the shortest response time. NetSolve provides the user with a "best effort" to find this *best* resource.

As seen on figure 2, a NetSolve client sends a request to an instance of the NetSolve agent. This instance of the agent has some knowledge about the computational resources in the system. Hopefully this knowledge is not too out of date (which is ensured by a set of protocols we do not have space to describe here) and, for each resource M, allows a fairly accurate computation of :

- T_n : the time to send the data to M and receive the result over the network, and
- T_c : the time to perform the computation on M.

All the details about the protocols involved in this computations are given in [17]. The whole idea behind this scheme is that it would be too inefficient to have the agent compute exact values for T_n and T_c for each incoming request. Instead, we prefer to have a quick estimate, which might not be as accurate.

Simultaneous resources :

Using the nonblocking interfaces to NetSolve, the user can design a NetSolve application that has some parallelism. Indeed, it is possible to send asynchronously several requests to NetSolve. The load balancing strategy described above insures that these problems will be solved on different machines, in parallel. The client has then just to wait for the results to come back.

Here is another strength of NetSolve : as soon as a new resource is started, it takes part in the system, and can be used. Therefore, without modifying his code or knowing in fact anything about the servers, a user can see the performance of his application greatly improved.

2.5 Fault Tolerance

Fault tolerance is an important issue in any loosely connected distributed system like NetSolve. The failure of one or more components of the system should not cause any catastrophic failure. Moreover, the number of side effects generated by such a failure should be as low as possible, and the system should minimize the drop in performance. We tried to make NetSolve as fault tolerant as possible.

A first aspect of fault-tolerance in NetSolve takes place at the server level. It is possible to stop a NetSolve server (resource or instance of the agent) at any time, and restart it safely at any time. In fact, every NetSolve server is an independent entity. This insures that the NetSolve system will remain coherent after any kind of network/machine problem. In the installation of NetSolve at the University of Tennessee, the whole system is managed by a 'cron' job, and servers are restarted automatically after machines go down for back-ups for instance.

Another aspect of fault tolerance is that it should minimize the side effects of failures. To this end, we designed the client-server protocol as follows. When the NetSolve agent receives a request for a problem to be solved, it sends back a list of computational servers sorted from the most to the least suitable one. The client tries all the servers in sequence until one accepts the problem. This strategy allows the client to avoid sending multiple requests to the agent for the same problem if some of the computational servers are stopped. If at the end of the list no server has been able to answer, the client asks another list from the agent. Since it has reported all the encountered failures, it will receive a different list.

Once the connection has been established with a computational server, there still is no guarantee that the problem will be solved. The computational process on the remote host may die for some reason. In that case, the failure is detected by the client, and the problem is sent to another available computational server. This process is transparent to the user but, of course, lengthens the execution time. The problem is migrated between the possible computational servers until it is solved or no server remains.

3 Conclusions and Future Work

The NHSE is providing a means for the HPCC community to share software and information and thus broaden and accelerate the use of high performance computing technologies in scientific and engineering applications. By supplying the tools and mechanisms for HPCC repositories to interoperate, the NHSE is enabling different HPCC agencies and research groups to leverage each others efforts. During the next year, the NHSE will be bringing online several new domain-specific repositories as well as promoting the review and evaluation of software in these domains.

The NetSolve project is still at an early development stage and there is room for improvement at the interface level as well as at the conceptual level. At the interface level, we are thinking of providing a Java interface to NetSolve. At the conceptual level, the load-balancing strategy must be improved in order to change the "best guess" into a "best choice" as much as possible. The challenge is to come close to a best choice without flooding the network. The danger is to waste more time computing this best choice than the computation would have taken in the case of a best guess only. All these improvements are intended to combine ease of use, generality and performance, the main purposes of the NetSolve project.

We plan to investigate extending the NHSE Repository in a Box toolkit with a remote execution facility based on NetSolve. This facility would allow repository maintainers to provide remote access to software, instead of having users download and install the software on their own systems. We will also investigate how to provide server safe execution environments for user code so that users may upload functions for execution on a remote server. This capability is important for software packages that require user-defined functions to be provided as input.

References

1. Shirley Browne, Jack Dongarra, Stan Green, Keith Moore, Tom Rowan, Reed Wade, Geoffrey Fox, Ken Hawick, Ken Kennedy, Jim Pool, Rick Stevens, Robert Olsen, and Terry Disz. The National HPCC Software Exchange. *IEEE Computational Science and Engineering*, 2(2):62–69, 1995.
2. Jack J. Dongarra and Eric Grosse. Distribution of mathematical software via electronic mail. *Communications of the ACM*, 30(5):403–407, May 1987.
3. Ronald F. Boisvert, Sally E. Howe, and David K. Kahaner. GAMS: A framework for the management of scientific software. *ACM Transactions on Mathematical Software*, 11(4):313–355, December 1985.
4. Shirley Browne, Jack Dongarra, Kay Hohn, and Tim Niesen. Software repository interoperability. Technical Report CS-96-329, University of Tennessee, 1996.
5. Inc The Math Works. *MATLAB Reference Guide*. 1992.
6. *Ninf : Network based Information Library for Globally High Performance Computing*. Proc. of Parallel Object-Oriented Methods and Applications (POOMA), Santa Fe, 1996.
7. W. Gander P. Arbenz and M. Oettli. The remote computational system. *Lecture Note in Computer Science, High-Performance Computation and Network*, 1067:662–667, 1996.
8. Sun Microsystems, Inc. XDR: External Data Representation Standard. RFC 1014, Sun Microsystems, Inc., June 1987.
9. D. Kincaid C. Lawson, R. Hanson and F. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software*, 5:308–325, 1979.
10. S. Hammarling J. Dongarra, J. Du Croz and R. Hanson. An extended set of fortran basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–32, 1988.
11. I. Duff J. Dongarra, J. Du Croz and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.
12. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM Philadelphia, Pennsylvania, 2 edition, 1995.
13. David M. Young David R. Kincaid, John R. Respess and Roger G. Grimes. Itpack 2c: A fortran package for solving large sparse linear systems by adaptive accelerated iterative methods. Technical report, University of Texas at Austin, Boeing Computer Services Company, 1996.
14. C. B. Moler J. J. Dongarra, J. R. Bunch and G. W. Stewart. *LINPACK Users' Guide*. SIAM Press, 1979.
15. A. Cline. Scalar- and planar-valued curve fitting using splines under tension. *Communications of the ACM*, 17:218–220, 1974.
16. J. Dongarra and D. Walker. Software libraries for linear algebra computations on high performance computers. *SIAM Review*, 37(2):151–180, 1995.
17. *NetSolve: A Network Server for Solving Computational Science Problems*. To appear in Proc. of Supercomputing '96, Pittsburgh, 1996.

This article was processed using the LaTeX macro package with LLNCS style