from *SIAM News*, Volume 31, Number 1

Network-Enabled Solvers And the NetSolve Project

By H. Casanova, J.J. Dongarra, and K. Moore

With the beginning of the 21st century will come new challenges for large-scale applications involving communication with, and coordination of, large numbers of geographically dispersed information sources supplying information to large numbers of geographically dispersed information consumers. Applications of this class will require an environment that supports long-term or continuous, reliable and fault-tolerant, highly distributed, heterogeneous, and scalable computing capability.

The characteristics of such applications include distributed data collection, distributed computation (often in significant amounts), distributed control, and distributed output. Many of these applications will require high levels of reliability and continuous operation, even if individual nodes or links may fail or otherwise be unavailable. Such applications will be constructed from a wide variety of computational components (including smart sensors, personal digital assistants, workstations, and supercomputers), and a wide variety of communications media (wire, optical fiber, terrestrial radio, satellite) with varying degrees of link reliability, bandwidth, and message loss.

APPLICATIONS ON ADVANCED ARCHITECTURE COMPUTERS
Greg Astfalk, Editor

The reliability requirement means that such applications must degrade gracefully rather than fail in the presence of node or link failures, insufficient communications bandwidth, or high message loss rates. Since some computational resources may not be available on a continuous basis, applications may have to adapt to varying amounts of computational power. Because of the potential for hostile attack on such systems, they must have a high degree of security, both for authentication of data and for maintaining the privacy of sensitive information.

To facilitate the construction of such systems, we need to develop new programming environments that integrate computation, data gathering, data storage, resource management, and human–computer interaction into a common framework. The framework should provide high availability and reliability through replication of both data and computational resources and through careful resource management. Such network-enabled programming environments will be based on a number of technologies currently being developed by various research groups.

As a result of advances in hardware, networking infrastructure, and algorithms, networked scientific computing can now be used to solve highly compute-intensive problems in many areas. In the networked computing paradigm, vital pieces of software and information used by and within a computing process are spread across the network. The requisite pieces are identified and linked together only at run time. In the current software usage model, by contrast, a copy (or copies) of task-specific, monolithic software is purchased for use within local hosts; it can even be distributed on a collection of local hosts. With networked computing, software is viewed no longer as a product, but rather as a service. The software developer provides a computing service to interested parties over the network. The raw computing power to run this service might be purchased from the software service provider, provided by the end user, or even purchased from a third-party computing service provider. The information needed to use this service might be available only from disparate sources. Clearly, the networked computing model does not apply to all computing services; basic operating-system and network-access software will probably be permanently resident on the user's machine.

One advantage of this paradigm is that the software provider no longer needs to release new versions and upgrades when software has been improved. The user simply sees improved service at the next run-time invocation. In this sense, the system is analogous to the phone system—changes in the software of a local switch are completely transparent to the user, except for the availability of additional or enhanced functionality. Similarly, the service provider can upgrade hardware without inconveniencing the user. We envision that this model will eventually become fully automated and effectively transparent to the user. It is the aim of the research described here to help achieve these goals in the context of scientific computing.

The user who decides to use freely available numerical software must first look for the appropriate library or set of routines needed for the specific computational problem. Usually, such libraries can be found in software repositories. One well-known repository is Netlib [1], which is maintained by several institutions and universities as a collaborative effort.

Software repositories present some intrinsic difficulties for the unexperienced user: They are generally very large, and they contain libraries of very different types. Once the appropriate software has been located, it must be downloaded and installed. Depending on the nature of the software, this step can be nontrivial, especially for a user not accustomed to a task of this kind. The biggest steps still remain—learning how to use the software and learning how to write a program in terms of the library components. These tasks can be formidable and time-consuming—and we have not even mentioned the debugging phase.

To address these challenges, we have developed a system called NetSolve. NetSolve provides the user with a pool of *computational resources*. These resources are computational servers that provide run-time access to arbitrary numerical libraries.

What is NetSolve?

NetSolve is a network-enabled solver-based system designed to solve computational science problems over a network. A number of interfaces have been incorporated within the NetSolve software. Users of C, Fortran, MATLAB, Java, or the Web can easily use the NetSolve system. Because the underlying computational software can be any scientific package, good performance results and great flexibility and extensibility are ensured. Moreover, NetSolve uses a load-balancing strategy to improve the use of the computational resources available.

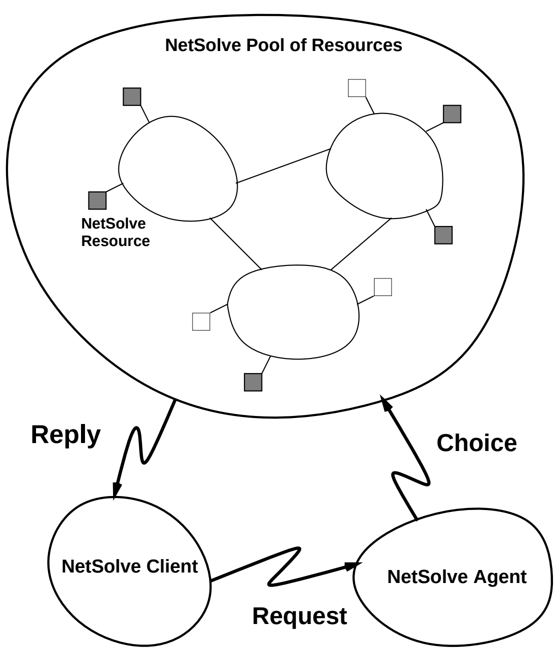

Figure 1. *The high-level organization of NetSolve.*

The Organization of NetSolve

We can distinguish three main paradigms for network-based systems: *proxy computing*, *code shipping*, and *remote computing*. These paradigms differ in the way they handle the user's data and the program that operates on these data. In *proxy computing* the data and the program reside on the user's machine; both are sent to a server that runs the code on the data and returns the result. In *code shipping* the program resides on the server and is downloaded to the user's machine; it operates on the data and generates the result on that machine. This is the paradigm used by Java applets within Web browsers. In the third paradigm, *remote computing*, the program resides on the server. The user's data are sent to the server, where the programs or numerical libraries operate on them; the result is then sent back to the user's machine. NetSolve uses the third paradigm; it is a client-server network-based system.

NetSolve provides the user with pools of computational resources. These resources are, in fact, *computational servers* that provide run-time access to arbitrary numerical libraries. The NetSolve computational servers have the following abilities:

- Uniform run-time access to the software. The servers give users the illusion that they have access to a uniform set of subroutines/ functions and provide direct access to computational modules.
- Configurability. The servers are not limited to the use of any particular software because they use a general framework that facilitates the integration of new functionalities. NetSolve can, at will, extend and encompass new numerical applications from any numerical library.
- Preinstallation. Numerical software is, of course, preinstalled on the servers' sites. The user is not responsible for installing or maintaining any numerical software at all.

To make the implementation of such a computational server model possible, we have designed a machine-independent, general way of describing a numerical computation. We have also designed a set of tools for generating new computational modules as easily as possible. The main component of this framework is a *descriptive language* that is used to describe each separate numerical functionality provided on a computational server. Files written in this language can be compiled by NetSolve into actual computational modules executable on any UNIX platform.

NetSolve also includes a Java applet that facilitates the generation of description files. The Java applet can be used by anyone on the Internet to create new computational resources. This framework also allows increased collaboration between research teams. Indeed, description files need to be generated only once and can be reused in a machine-independent manner to set up new computational resources anywhere on the Net. So far, such description files have been written for the following numerical libraries: FitPack, ItPack, MinPack, FFTPACK, LAPACK, BLAS, and QMR. NetSolve computational servers providing access to these packages are running on a 24-hour basis at the University of Tennessee and at other locations.

Any of the NetSolve client interfaces can be used to send requests to the NetSolve computational servers. User requests are not sent directly to the computational resources, however, but rather are processed by another component of the system, a NetSolve *agent*. The agent decides on the computational server to which a user's request will be assigned. Thus, the agent is really the mastermind behind the NetSolve strategy, and the efficiency of the system depends entirely on its decisions. Figure 1 shows this organization.

One of the roles of the NetSolve agent is to perform load balancing among the different computational resources. NetSolve is inherently a multirequest system.

Several users can compete for resources by contacting the agent(s)—the same or others—managing the same pool of resources. Alternatively, a single user can send multiple asynchronous requests at once (as discussed in the description of the user interfaces). For each incoming request, the NetSolve agent chooses a computational server where the numerical computation will be performed.

For each server, the agent can use information contained in the user request (e.g., type of computation, size of the problem), static information about the server (e.g., speed of the host, numerical server available), predictions about the workload of the server's host, and the distance to the server's host over the network. These different pieces of information are then combined to obtain an estimate of the time required to process the user request on each computational server, including network time and CPU time. For each request, the NetSolve agent sorts the appropriate computational servers according to these estimated times and processes the request accordingly. More details on this strategy can be found in [2].

Where Can NetSolve Be Used?

The different hosts that participate in the NetSolve protocol can be anywhere on the Internet. In fact, they can be administered by different institutions. NetSolve does not assume any centralized control over the hosts in the system. On the contrary, each process (computational server or agent) is an independent entity: It can be safely stopped and restarted at any time, without jeopardizing the integrity of the system. The flexibility of this approach does require that NetSolve implement some kind of fault-tolerance mechanisms. Indeed, any resource can become unreachable at any moment, perhaps because of a network failure, a host failure, or simply a system administrator rebooting a host.

NetSolve also can be used on an intranet, inside a research department or a university, without participating in any Internet computation. Even though such a setting is more stable than an Internet-based NetSolve configuration, fault tolerance is still required.

Currently, NetSolve uses the following strategy for fault tolerance. The NetSolve system ensures that a user request will be completed unless every single resource has failed. When a client sends a request to a NetSolve agent, it receives a sorted list of computational servers to try. When one of these servers has been successfully contacted, the numerical computation starts.

If the contacted server fails during the computation, another server is contacted and the computation is restarted. This whole process is transparent to the user. If all the servers fail, the user is notified that the computation cannot be performed at that time. This simple fault-tolerant approach will be improved in a future version of the NetSolve software.

What Interfaces Does NetSolve Provide?

One of our major concerns in designing NetSolve was to provide several interfaces so that a wide range of users would be targeted. Currently, NetSolve provides Application Program Interfaces (APIs) as well as higher-level interfaces. C, Fortran, and Java APIs are already available, as are a MATLAB interface and a graphical Java interface. Another concern was keeping the interfaces as simple as possible. For example, the MATLAB interface contains only two functions that allow users to submit problems to the NetSolve system.

In addition to traditional synchronous calls, every interface provides asynchronous calls to NetSolve. When several asynchronous requests are sent to a NetSolve agent, they are dispatched among the available computational resources according to the load-balancing schemes implemented by the agent. Hence, the user—with virtually no effort—can achieve coarse-grained parallelism either from a program or from interaction with a high-level interface. All the interfaces are described in detail in [3].

MATLAB Interface Example

The MATLAB interface to NetSolve is easy to use and is no different from any other MATLAB extension. With this interface, the user can find out what resources, both hardware and software, are available in the NetSolve system. Consider a user who wants to perform two independent computations, for instance a vector sort and an eigenvalue problem. After typing `netsolve` at the MATLAB prompt and browsing the list of resources, the user finds out that the two *problems* to be used are `qsort` and `eig`. The following step-by-step procedure illustrates the use of NetSolve from MATLAB.

First, the data must be loaded from disk into memory:

```
>> load a
>> load v
```

The user then sends the first asynchronous request to NetSolve to perform the eigenvalue computation:

```
>> [request1] = netsolve_nb( 'send' , 'eig' , a )
   NetSolve : contacting server 'ig.cs.utk.edu'
   request1 = 0
```

NetSolve informs the user that the problem is being performed on the machine `ig.cs.utk.edu` and returns a request handler in the form of an integer (`0`). The request for the vector sort follows:

```
>> [request2] = netsolve_nb( 'send' , 'qsort' , v )
    // sends the request for the vector sort
    contacting server 'cupid.cs.utk.edu'
    request2 = 1
```

At this point, the two requests are being serviced in parallel on two different servers. The user can now use MATLAB to perform local computations. When the user deems it appropriate, the results of the computation can be checked as follows:

```
>> [eigenvalues] = netsolve_nb( ‘probe’ , request1 )
   NetSolve : Not ready yet
>> [sorted] = netsolve_nb( ‘wait’ , request2 )

   sorted = [-1.23 ....]
```

The user first *probed* for the eigenvalues and was informed that they were not yet available. The user then *waited* for the sorted vector to be computed. The sorted vector is returned in the variable `sorted` when the call returns. Finally, the user can wait for the eigenvalues to be available:

```
>> [eigenvalues] = netsolve_nb( ‘probe’ , request1 )

   eigenvalues = [0.23+i23.11 ....]
```

This example shows clearly that by using NetSolve from MATLAB, the user has the ability not only to access a variety of computational resources, but also to achieve parallelism at almost no cost. The NetSolve MATLAB interface has other functionalities that are not part of this short example. The complete description of the interface, along with a complete example, can be found in [3].

Fortran Interface Example

Another important aspect of NetSolve is that the computational servers can enforce any arbitrary calling sequence from C and Fortran to the problems they handle. It is then possible to have the calls from the NetSolve client exactly match those that would be made directly to the underlying numerical libraries. The following Fortran example illuminates this aspect:

```
      parameter( MAX = 100)
      double precision A(MAX,MAX),B(MAX)
      integer IPIV(MAX),N,INFO,LWORK
      integer NSINFO
C
C     *************************
C     * Direct call to LAPACK *
C     *************************
C
      call DGESV(N,1,A,MAX,IPIV,B,MAX,INFO)
      ...
C     *************************
C     * Call to NetSolve      *
C     *************************
C
      call NETSL(‘DGESV()’,NSINFO,N,1,A,MAX,IPIV,B,MAX,INFO)
```

With the exception of the first two arguments, the call to NetSolve matches the call to LAPACK. The first two arguments are the *problem name* and the NetSolve error code. It is clear that a user who is accustomed to a particular numerical library can easily switch to NetSolve. Existing codes can then be converted using asynchronous calls to NetSolve, yielding some degree of parallelism at almost no extra cost to the user.

Obtaining NetSolve

Currently, we have released version 1.0 of NetSolve (both clients and servers). The NetSolve home page, which is located at http://www.cs.utk.edu/netsolve, contains detailed information and source code.

To allow users to try out NetSolve as soon as they download the client distribution, we maintain a pool of computational servers at the University of Tennessee (as well as in some other sites). These servers can solve various numerical problems in several fields, including linear algebra, fast Fourier transform, optimization, and curve fitting. The numerical functionalities are constantly increasing, and new servers are being started as the demand increases.

NetSolve is a continuing project, and several research issues are under investigation for the next release. One of the main improvements that we would like to make to the paradigm is to allow dynamic software–hardware computational resource binding. In the present version of the software, computational servers are given access to computational software and started on a host. New numerical functionalities can be added to the server, but this decision has to be made by the NetSolve administrator.

We envision a system where a server (or agent), upon receiving a request for an unknown numerical computation, could contact a well-established software repository and download the appropriate code to perform the computation. Software resources, hardware resources, and data resources could be dynamically bound yet transparent to the user. The Netlib

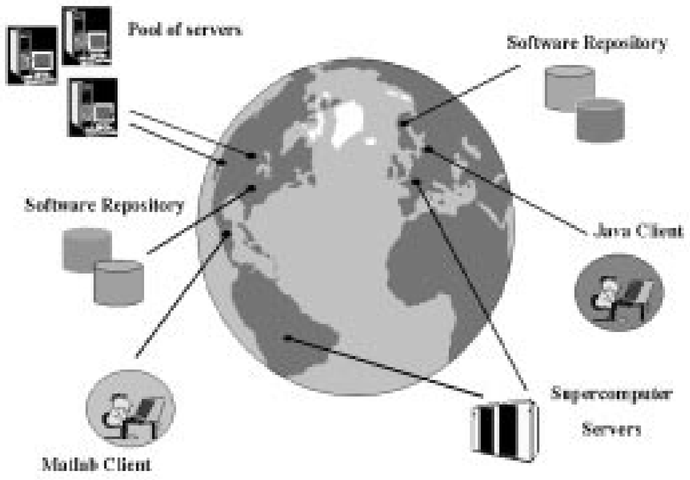

Figure 2. *NetSolve-enabling global collaboration.*

repository seems to be the natural choice for this revolutionary system. This new paradigm will require consideration of several issues, such as security, software caching mechanisms, and software authentication. Success in this endeavor, however, would represent a breakthrough in global metacomputing and collaboration, as depicted in Figure 2.

References

[1] S. Browne, J. Dongarra, E. Grosse, and T. Rowan, *The Netlib Mathematical Software Repository*, D-Lib Magazine, September 1995, http://www.cnri.reston.va.us/home/dlib/september95/09contents.html.

[2] H. Casanova and J. Dongarra, *NetSolve: A network-enabled server for solving computational science problems*, Int. J. Supercomput. Appl. and High Performance Comput., 11–3 (1997), 212–223, http://www.netlib.org/utk/people/JackDongarra/PAPERS/netsolve.ps.

[3] H. Casanova, J. Dongarra, and K. Seymour, *Client User's Guide to NetSolve*, Technical Report CS–96–343, Department of Computer Science, University of Tennessee, Knoxville, 1996, http://www.cs.utk.edu/~library/TechReports/1996/ut-cs-96-343.ps.Z.

Henri Casanova (casanova@cs.utk.edu) is a graduate research assistant at the University of Tennessee, Knoxville. Jack Dongarra (dongarra@cs.utk.edu) is a Distinguished Professor at the University of Tennessee, Knoxville, and a Distinguished Scientist at Oak Ridge National Laboratory. Keith Moore (moore@cs.utk.edu) is a research associate at the University of Tennessee, Knoxville.