Network-Enabled Solvers and the NetSolve Project

H. Casanova
J.J. Dongarra
K. Moore

The beginning of the 21st century will present new challenges for large-scale applications involving communication with, and coordination of, large numbers of geographically dispersed information sources, supplying information to a large number of geographically dispersed information consumers. Applications of this class will require an environment which 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 reliability and continuous operation, even though individual nodes or links may fail or otherwise be unavailable. Such applications will be constructed out of 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. The reliability requirement means that such applications must degrade gracefully rather than fail in the presence of node or link failures, or with insufficient communications bandwidth and high message loss rates. Since some computational resources may not be available on a continuous basis, applications may have to adapt to varying computational power. The potential for hostile attack to such systems requires that they have a high degree of security, both for authentication of data and privacy of sensitive information.

To facilitate the construction of such systems, development of new programming environments which integrate computational, data gathering, data storage, resource management, and human-computer interaction into a common framework are needed. The framework should provide high availability and reliability through replication of both data and computational resources and by careful resource management. Such a network-enabled programming environment is based on a number technologies currently being developed by various research groups.

Enabled by advances in hardware, networking infrastructure and algorithms, highly compute intensive problems in many areas can now be successfully solved using networked scientific computing. 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. This is in contrast to the current software usage model where a copy (or copies) of a task-specific, monolithic software is purchased for use within local hosts. It may even be distributed on a collection of local hosts. With networked computing, the view of software changes from a product to a service. The software developer provides a computing service to interested parties over the network. The raw computing power to run this service may 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 machine. One advantage of this paradigm is that as software is improved by the software provider, there is no need to release new versions and upgrades. The user simply sees an improved service at the next run-time invocation. The analogy could be to the phone system—changes in the software of the local switch are completely transparent to the user, except for the availability of additional or enhanced functionality. Similarly, the service provider can upgrade the hardware without affecting the user. We envision that this model will eventually become fully automated and effectively transparent to the user. It is the aim of this research to facilitate this 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. A specific well known repository is Netlib [1]. Netlib is maintained through the collaborative effort of several institutions and universities. Software repositories present some intrinsic difficulties for the unexperienced user: they are generally very large, and they contain very different types of libraries. Once the appropriate software has been located, it must be downloaded and installed. Depending on the nature of the software, this step might be nontrivial, especially for a user not used to this kind of task. 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. Further, we have not even mentioned the debugging phase.

To implement a system to address the challenges highlighted above we have developed 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 different interfaces have been incorporated within the NetSolve software. Users of C, Fortran, MATLAB, Java, or the Web can easily use the NetSolve system. The underlying computational software can be any scientific package, thereby ensuring good performance results and great flexibility and extensibility. Moreover, NetSolve uses a load-balancing strategy to improve the use of the computational resources available.

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 this data. In *proxy computing* the data and the program reside on the user's machine and are both 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 where 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 is sent to the server, where the programs or numerical libraries operate on it; 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 using any particular software because they use a general framework to integrate new functionalities easily. NetSolve can extend and encompass new numerical applications at will, 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 to generate 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 to easily generate 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.

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

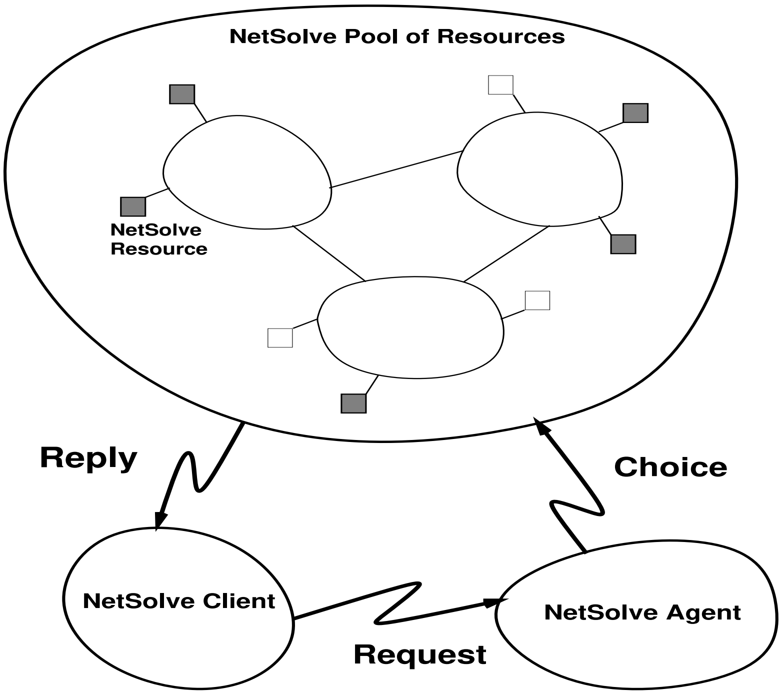

Figure 1: The high-level organization of NetSolve.

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 the resources by contacting the same, or different, agent(s) managing the same pool of resources. Alternatively, a single user can send multiple asynchronous requests at once (as we will see 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 administrated by different institutions. NetSolve does not assume any centralized control over the different hosts in the system. On the contrary, each process (computational server or agent) is an independent entity: it can safely be 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 this time. This simple fault-tolerant approach will be improved in a future version of the NetSolve software.

What interfaces does NetSolve provide?

A major concern in designing NetSolve was to provide several interfaces in order to target a wide range of users. Currently, NetSolve provides Application Program Interfaces (APIs) as well as higher-level interfaces. C, Fortran, and Java APIs are already available, as well as 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. Every interface provides asynchronous calls to NetSolve in addition to traditional synchronous calls. 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 from either 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 form any other MATLAB extension. Using this interface, the user can find out what resources are available in the NetSolve system, including hardware and software resources. 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* he wants to use are `qsort` and `eig`. We show here step by step what the user is supposed to do to use NetSolve from MATLAB.

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

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

Then, the user send 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 his 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 request are being serviced in parallel on two different servers. The user can now perform local computations using MATLAB. When he deems it appropriate, he can check the results of the computation 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. Then the user *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 ....]
```

¿From this example it is easy to see how using NetSolve from MATLAB provides the ability not only to access a variety of computational resources, but also to achieve parallelism at almost no cost for the user. There are other functionalities in the NetSolve MATLAB interface that are not part of this short example. The complete description of the interface is found in [3], as well as a complete example.

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 problem they handle. It is then possible to have the calls from the NetSolve client exactly match the ones that would be made directly to the underlying numerical libraries. The following Fortran examples 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 then easily seen that a user who is accustomed to a particular numerical library can switch to NetSolve easily. Existing codes can then be converted, using asynchronous calls to NetSolve and 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 contains detailed information and source code. The home page is located at `http://www.cs.utk.edu/netsolve`.

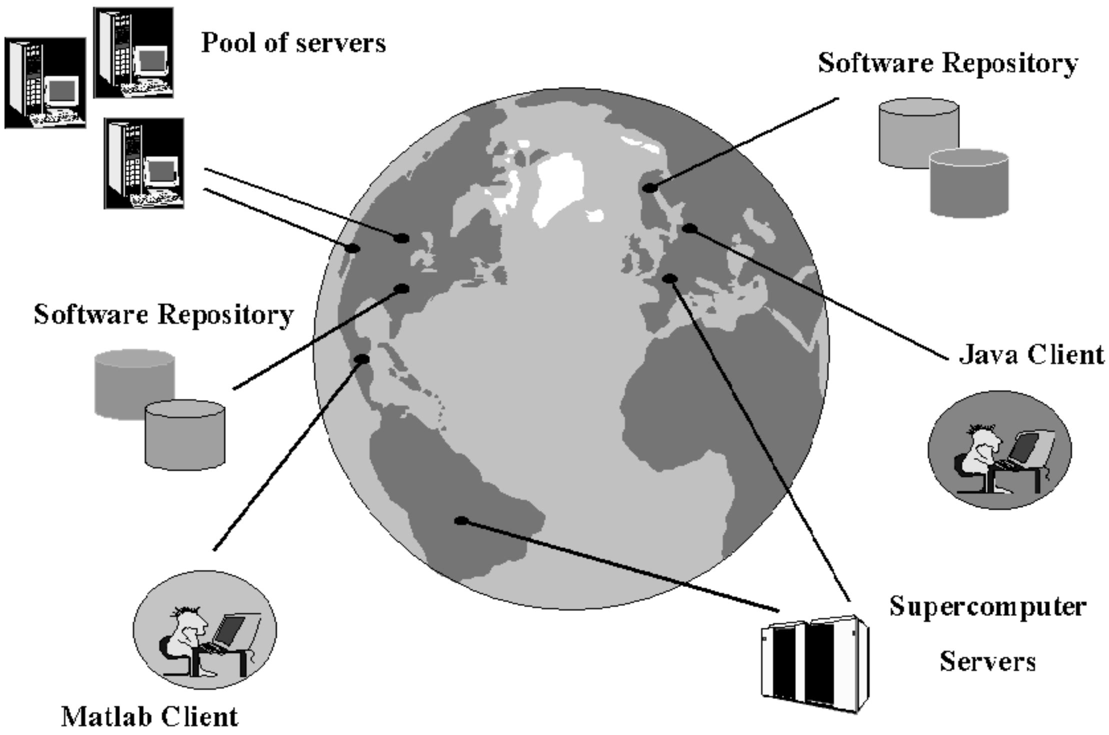

Figure 2: NetSolve-enabling global collaboration

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

References

[1] S. Browne and J. Dongarra and 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 , The International Journal of Supercomputer Applications and High Performance Computing.* The International Journal of Supercomputer Applications and High Performance Computing, Volume 11, Number 3, pp 212-223, Fall 1997, *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, TN, 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 the Oak Ridge National Laboratory.

Keith Moore (`moore@cs.utk.edu`) is a Research Associate at the University of Tennessee, Knoxville.