Applying NetSolve's Network-Enabled Server

HENRI CASANOVA
University of Tennessee
JACK DONGARRA
University of Tennessee and Oak Ridge National Laboratory

The scientific community has long used the Internet for communication of e-mail, software, and papers. Until recently, there has been little use of the network for actual computations. This situation is changing rapidly and will have an enormous impact on the future. The NetSolve system described here has a significant role to play in these developments.

The NetSolve project, underway at the University of Tennessee and Oak Ridge National Laboratory, lets users access computational resources, both hardware and software, distributed across the network. Thanks to a variety of interfaces, users can easily perform scientific computing tasks without having any computing resources installed on their computers. Research issues involved in the NetSolve system include fault tolerance, load balancing, user-interface design, computational servers, virtual libraries, and network-based computing. As the project matures, several promising extensions and applications of NetSolve will emerge.

In this article, we describe the project and examine some of the extensions being developed for NetSolve: an interface to the Condor system, an interface to the ScaLapack parallel library, a bridge with the Ninf system, and an integration of NetSolve and ImageVision.

Competing paradigms

Ongoing advances in hardware, networking infrastructure, and algorithms let us successfully attack a wide range of computationally intensive problems using networked, scientific computing. The networked-computing paradigm spreads vital pieces of software and information used by a computing process across the network, identifying and linking them only at runtime. By contrast, in the current software usage model, users acquire a copy (or copies) of task-specific software packages for use on local hosts.

There are three main paradigms for such networked computing paradigms: *proxy computing*, *code shipping*, and *remote computing*. These paradigms differ in the way they handle the user's data and in the program that operates on this data. In proxy computing, the data and the program reside on the user's machine; both travel 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 Java applets use within Web browsers. In remote computing, the program resides on the server. The user's data travels to the server, where the programs or numerical libraries operate on it; the result then returns to the user's machine.

The NetSolve system uses this third paradigm (see Figure 1). NetSolve provides the user with a pool of computational resources. These resources are computational servers that have access to ready-to-use numerical software. As the figure shows, the computational servers can be running on single workstations, networks

1070-9924/98/$10.00 © 1998 IEEE

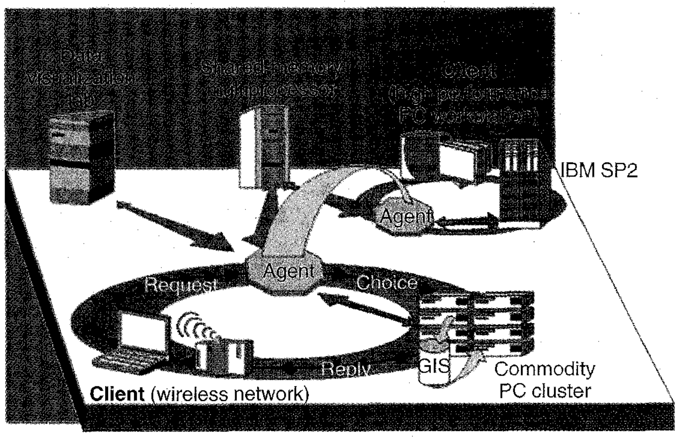
Data visualization lab
Shared-memory multiprocessor
Client (high performance PC workstation)
IBM SP2
Agent
Agent
Request
Choice
Reply
GIS
Commodity PC cluster
Client (wireless network)

Figure 1. NetSolve's organization.

of workstations that can collaborate for solving a problem, or massively parallel processor systems. Using one of the NetSolve client interfaces, users can send requests to the NetSolve system, asking that one of the servers carry out their numerical computation. As its main role, the NetSolve agent processes this request and chooses the most suitable server for this particular computation. Once the agent chooses a server, that server is assigned the computation, uses its available numerical software, and eventually returns the results to the user.

As Figure 1 shows, there can be multiple instances of the NetSolve agent on the network, and different clients can contact different agents depending on their locations. The agents can exchange information about their different servers and allow access from any client to any server, if desirable. Suppose, for example, that the set of computational resources spans several local-area networks and that users on each network want to use NetSolve to perform scientific computations. A NetSolve agent can be started on each network, so user requests always go to the "closest" agent for processing. NetSolve can operate either via the Internet or on an intranet, such as inside a research department or a university, without participating in any Internet-based computation. Also, NetSolve's system configuration is entirely flexible: any server or agent can stop and start or restart at any time without jeopardizing the system's integrity.

The NetSolve agent is the primary participant in the management of the different computational resources (hardware and software) as it is in charge of load balancing and fault tolerance.

The computational resources

One challenge in building the NetSolve system was to design a suitable model for the computational servers. For users to invoke numerical software directly through our servers, three major features seemed to emerge as mandatory for the servers:

- *Uniform access to the software*: The servers should give users the illusion that they have access to a uniform set of subroutines and functions. This is a critical point because we want to hide, as much as possible, the specifics of the underlying numerical softwares. In this way, users need not undergo long learning phases when using a new set of functions.
- *Configurability*: The servers should not be limited to any particular software. We therefore needed to provide a framework to easily add functionality to a computational server. This would let the system extend and encompass new numerical applications at will.
- *Preinstallation*: Of course, the user should not be responsible for installing any numerical software directly. The numerical software available through the servers should be ready to use and already compiled to the target architecture. Or, more generally, the system could dynamically handle installation and compilation itself, without any intervention from the user.

The current design. For implementing such a computational server, we have designed a general, machine-independent way of describing a numerical computation, as well as a set of tools to easily generate new computational modules. This framework's main component is a descriptive language used to describe each separate numerical functionality of a computational server. NetSolve can compile the description files written in this language into actual computational modules executable on any Unix or NT platform. These files are really NetSolve wrappers around the scientific libraries and should not contain any numerical code.

The NetSolve problem description contains information about what numerical-library function to use, what input to expect, and what output will be generated. The file must also contain an estimate of the problem's algorithmic complexity as a function of the input size. Such an estimate can be well known (such as for direct solvers) or can depend on the input data itself (such as for iterative solvers). The latter case is still an open question in NetSolve.

This approach offers several advantages. Machine independence is one, as is the ability to integrate arbitrary software components into NetSolve. But this framework also allows increased collaboration between research teams and institutions. Indeed, description files for a given numerical library need to be written only once. Any institution wanting to set up servers can then exchange these files. Each time a new description file is created, the capabilities of the entire NetSolve system increase.

These advantages, however, are effective only if the process of creating new problems and adding them to a computational server is reasonably straightforward. For this reason, we developed a graphical user interface to handle description-file generation. The interface performs various error checks on the user input, which mostly consists of mouse clicks and choices in menus. Using the interface is much easier than creating a description file manually, especially as the sophistication of the problem increases.

Ultimately, we want to have a NetSolve description file repository on the Web. From such a repository, users could download description files at will to set up computational servers. The actual numerical software should also be available to make the creation of these servers nearly immediate.

Existing resources. The NetSolve team has generated a number of description files for numerical libraries that cover several fields of computational science—linear algebra, optimization, and fast Fourier transforms, for example. NetSolve computational servers providing access to these libraries are currently running at the University of Tennessee and at other locations worldwide. NetSolve users have also developed their own description files to start servers that answer their particular needs. For real-time information on the running servers, check the NetSolve Web page at *http://www.cs.utk.edu/netsolve/*.

The client interfaces

In designing NetSolve, we were most interested in providing several interfaces for a wide range of users. Users can invoke NetSolve through C, Fortran, Java, as well as on Matlab.[1] In addition, there is a Web-based Java GUI that lets users solve problems remotely. We also wanted to keep the interfaces as simple as possible. For example, there are only two calls in the Matlab interface, and they are sufficient to let users submit problems to the NetSolve system. Each interface provides asynchronous calls to NetSolve in addition to traditional synchronous or blocking calls. When several asynchronous requests travel 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 C or Fortran program, or from interaction with a high-level interface. User requests never contain numeric problem-solving code, but rather ask to use code that is preexisting on the NetSolve servers. (Our *Client User's Guide to NetSolve* describes all the interfaces.[2])

In designing NetSolve, we were most interested in providing several interfaces for a wide range of users. Users can invoke NetSolve through C, Fortran, Java, as well as on Matlab.

How the NetSolve agent works

The NetSolve agent operates both as a database and as a resource broker. The agent keeps track of information about all the servers in its resource pool, including their availability, load, network accessibility, and the range of computational tasks that they can perform. The agent then selects a server to perform the task, and the server responds to the client's request.

The agent as a database. Keeping track of what software resources are available and on which servers they are located is perhaps the NetSolve agent's most fundamental task. Because the computational servers use the same framework to contribute software to the system, the agent can maintain a database of different numerical functionalities available to the users. The protocol is fairly straightforward. Each time a new server starts, it sends an application request to an instance of the NetSolve agent. This request contains general information about the server (including its location), as well as the list of numerical functions it intends to contribute to the system. Eventually, the server is integrated into the system and can answer user requests.

The agent as a resource broker. The NetSolve agent's goal is to choose the best-suited computational server for each incoming request to the system. For each user request, the agent determines the set of servers that can handle the computation and ranks those servers from most suit-

An Interface to the Condor System

Condor, developed at the University of Wisconsin, Madison, is a high-throughput computing environment that can manage very large collections of distributively owned workstations.[1–3] Its development has been motivated by the ever-increasing need for scientists and engineers to exploit the capacity of such collections, mainly by taking advantage of otherwise wasted CPU cycles. Workstation owners individually define the conditions under which the workstation can be allocated by Condor to an external user. Condor jobs that consist of a single process are automatically checkpointed and migrated between workstations as needed to ensure eventual completion.

Condor's software architecture

A Condor pool consists of any number of machines, possibly of different architectures and operating systems that are connected by a network. Condor daemons constantly monitor the status of the individual computers in the cluster (the master daemon maintains the coherency of the set of daemons). Two daemons run on each machine, the *startd* and the *schedd.* The startd monitors information about the machine itself (load and mouse and keyboard activity, for example) and decides if it is available to run a Condor job. The schedd keeps track of all the Condor jobs submitted to the machine. One machine, the central manager, keeps track of all the resources and jobs in the pool. All the daemons report their information to a daemon (called the *collector*) running on the *central manager.* Finally, an additional daemon (the negociator) on the central manager periodically takes information from the collector to find idle machines and match them with waiting jobs. When a job is submitted to Condor, the scheduler on the central manager matches a machine in the Condor pool to that job. Once the job has begun, it is periodically checkpointed. It then can be interrupted and migrated within the Condor pool until completion. Figure A partly depicts this organization. Interfacing NetSolve and Condor makes it possible to give priority to the local users and provide only underutilized CPU cycles to NetSolve users.

A Condor pool as a NetSolve resource

Figure A shows how we can view an entire Condor pool as a single NetSolve computational resource. The Condor pool consists of several machines. The central manager at the top of the figure runs two daemons in addition to the usual startd and schedd: the negociator and the collector. All the other machines run the startd and the schedd daemons only. One machine also runs a customized version of the NetSolve server. When this server receives a request from a client, instead of creating a local child process running a computational module, it uses the Condor tools to submit that module to the Condor pool. The negociator on the central manager then chooses a target machine and notifies the startd of that machine to spawn the computational module. Due to fluctuations in the state of the pool, the computational module can then migrate among the pool's machines. When the results of the numerical computation are obtained, the NetSolve server transmits that result back to the client.

able to least. The agent is a resource broker in that once it returns the list of ranked computational servers to the client, it is not involved in the computation itself.

To perform the ranking, the agent uses computation-specific and resource-specific information. Computation-specific information is mostly included in the user request: size in bytes of the input data, size of the problem to be solved (such as dimensions of the matrices for a linear algebra computation), and so forth. Resource-specific information is composed of static and dynamic data. Each server communicates static system-specific data to the agent when the server first starts and is accepted in the system. This data mainly contains the server's host processor speed, the number of processors, and the complexity of the algorithms used by its numerical software. Dynamic data represents the load of the server's host, the network delays, and transmission rates to contact that host.[3]

Fault tolerance. As we've said, different institutions can administer the hosts in the NetSolve system. That's why NetSolve does not try to impose any control on the different resources. Indeed, we have said earlier that any NetSolve server can be stopped at any time (either willingly or because of a network/host/software failure). Although this approach is flexible, it requires NetSolve to implement some kind of fault-tolerance mechanisms.

The NetSolve system ensures that a user request will be completed unless every single resource capable of servicing the request has failed. When a client sends a request to a NetSolve agent, it receives a sorted list of computational servers to try. When the client has succeeded in contacting one of these servers, the numerical computation starts. If the contacted server fails during the computation, then another server is contacted and the computation restarts. In the current implementation, all input data resides on

A major issue still needs to be addressed: how does the NetSolve agent perceive a Condor pool as a resource? Indeed, the agent uses some performance metrics to assign client requests to computational resources. It is currently unclear how these metrics can be directly applied. As a possible approach, the NetSolve server running in the Condor pool could collect statistics on the pool behavior. The NetSolve agent could then use these statistics to compute predictions of job execution times in that pool. Finding the appropriate prediction technique will be the focus of the next step in NetSolve-Condor collaboration.

References

1. M. Litzkow, M. Livny, and M.W. Mutka, "Condor—A Hunter of Idle Workstations," *Proc. Eighth Int'l Conf. Distributed Computing Systems,* IEEE Computer Society Press, Los Alamitos, Calif., 1988, pp. 104–111.
2. M. Litzkow and M. Livny, "Experience with the Condor Distributed Batch System," *Proc. IEEE Workshop on Experimental Distributed Systems,* IEEE CS Press, 1990.
3. J. Pruyne and M. Livny, "A Worldwide Flock of Condors: Load Sharing among Workstation Clusters," *J. Future Generations of Computer Systems,* Vol. 12, 1996.

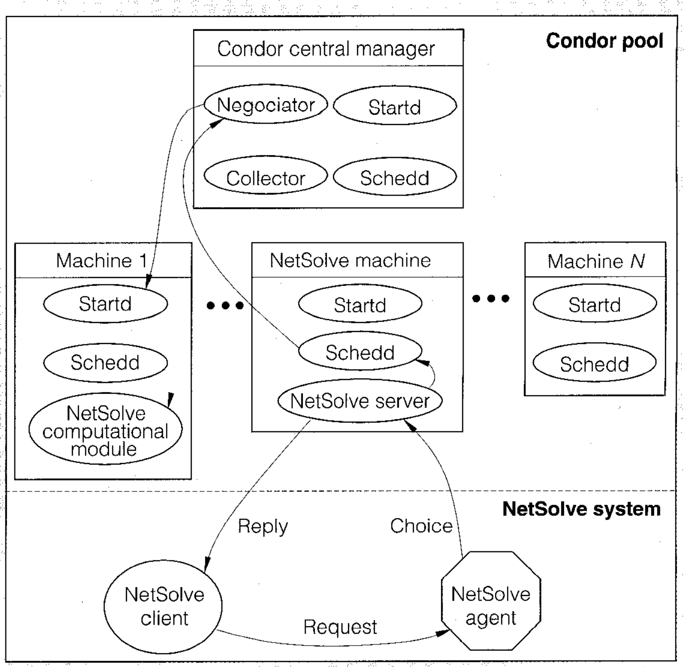

Figure A. NetSolve and Condor.

the user's machine during the computation and can then travel to another server after a failure. Future implementations will hand off the input data to NetSolve storage servers, where it will stay until the computation has finished (after possible failures and restarts). The whole restart process is transparent to the user.

This fault-tolerance mechanism is rather primitive (even though effective), and we are investigating techniques to allow task-checkpointing among servers. A first step is the Condor interface we discuss in the "An Interface to the Condor System" sidebar.

Using NetSolve

As an example of NetSolve's capabilities, let's look at two applications, once in neuroscience and another in performing simple parallelism with Matlab.

Neuroscience. The kind of network computing enabled by NetSolve is particularly appropriate for applications where the code is much larger than the data (such as PDE solvers, optimization, and symbolic mathematics). MCell, developed at the Computational Neurobiology Lab of the Salk Institute, is one such application.[4] This software performs 3D Monte Carlo simulations of cellular microphysiology for biologists and biochemists. MCell simulations take a few (typically small) files as input and produce a few small output files, making MCell an ideal application for NetSolve.

At the moment, MCell users manually start MCell jobs on different machines, where they have made the input files accessible, and manually gather the output files. This is extremely inconvenient, especially because they would like to run hundreds of computations in parallel on machines that are not on the same network file server. Furthermore, there is no fault-tolerance or load-balancing mechanism, both mandatory for such a large number of tasks.

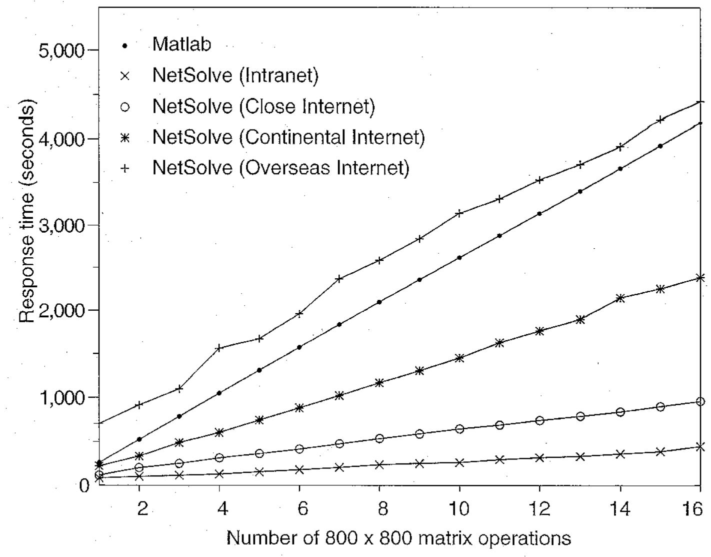

Figure 2. Multiple 800 x 800 matrix multiplications. The Intranet, Close Internet, Continental Internet, and Overseas Internet curves are for a client located respectively at the University of Tennessee; the Oak Ridge National Laboratory; the University of California, Berkeley; and the Danish Technical University in Denmark.

We have written a prototype NetSolve driver for MCell. The user passes (via a script) the list of all the MCell simulations to be performed, and the driver submits all the requests asynchronously to NetSolve. When the driver returns, all the output files have been created and are available to the user. The NetSolve agent ensures that the load balancing is correct and that the execution was fault tolerant. Our first experiment consisted in running the driver at the Salk Institute in La Jolla, California, and having 10 NetSolve MCell servers in Tennessee. We are working closely with the MCell team to modify MCell so that it calls NetSolve internally.

Simple parallelism from Matlab. In this example, the user is using Matlab on a Sun Sparc 5 workstation to compute the square of several matrices. In the experiment, matrices are 800 × 800, and the user wants to perform up to 16 computations. Seven NetSolve servers are available on Sun Ultra 1s.

Figure 2 shows the total execution times for various numbers of computations when the user is using either Matlab only or NetSolve via Matlab. When the user is using NetSolve, we have performed different measurements for different geographical locations of the client. The servers are located at the University of Tennessee.

When Matlab is used directly, the total execution time increases linearly with the number of operations because there is no parallelism. Using NetSolve improves performance considerably, except when the client is overseas. In fact, transferring the data and the result overseas is much more costly than performing a matrix square. In all the other cases, it is always better to use NetSolve, even if only one square is needed.

We can conduct the same experiment with any NetSolve client interface, including the Java GUI. Figure 3 shows sketches of the Matlab script that we used in the experiment. The first part of the script is pure Matlab, and the second part uses NetSolve and handles the task parallelism.

MultiMatlab,[5] developed at the Cornell Theory Center, could be used to run such a parallel Matlab script. However, MultiMatlab provides a low-level MPI-like interface and would not give the user the same service abstraction as NetSolve does. We could then write a layer on top of MultiMatlab that would create such an abstraction and implement fault-tolerance and load-balancing mechanisms, namely rewriting NetSolve with MultiMatlab. However, this approach would be very limited because users could call NetSolve only from Matlab, and Matlab is not freely distributed.

Related grid-oriented software

Both Ninf[6] and the Network-Enabled Optimization Server[7] bear similarities to NetSolve, but the differences are significant. NEOS addresses a specific field of computational science (optimization), whereas NetSolve can integrate virtually any processing of user data. Also, NetSolve's software architecture lets it be deployed on any scale with great flexibility, whereas NEOS is centralized at the Argonne National Laboratory.

Other than Condor, which we discuss in the sidebar, the two leading middleware systems for creating computational grids are Globus[8] and Legion.[9] They address somewhat different audiences. NetSolve targets any scientist or engineer, providing them with a high-level service. By contrast, both Globus and Legion are built on their own lower-level directory and communication services, making them significantly more elaborate to deploy.

Current and future directions

As NetSolve increases both in number of users and resources, maintaining a coherent resource space will become increasingly difficult. The issue of a robust and flexible naming strategy will undoubtedly arise. Several naming services have been designed (LDAP[10] and RCDS[11]) and implementations are starting to become available. Such services would provide a good basis for a metacomputing project such as NetSolve and would prevent the NetSolve developers from taking naming responsibilities.

The NetSolve model can accommodate both Globus and Legion (as agents and servers). As these systems gain maturity, we intend to review the current NetSolve design and implementation and gradually integrate new components from these systems. We have already integrated the Globus Heart Beat Monitor[12] into NetSolve to perform failure detection. Analogous investigations of the integration of NetSolve with Legion's approach to grid computing are also underway, which would provide robust mechanisms for security, data encryption and compression, and user-access control.

We are also investigating ways of producing a more robust implementation on top of distributed objects, for instance, with Corba[13] or possibly Java with RMI.[14] NetSolve provides a much higher level of abstraction (service versus object) to the user than either Corba or RMI, but could implement this abstraction along with load-balancing and fault-tolerance mechanisms in a distributed-object environment.

Finally, as the types of hardware resources and the types of numerical software available on the computational servers become more and more diverse, the resource broker embedded in the agent will need to become increasingly sophisticated, leading to many open research questions.

```
% loading the matrices
load 'mat1';...;load 'mat16';
Matfiles = {mat1,...,mat16};
clear 'mat1';...;clear 'mat16';
nb = 16;

% PURE MATLAB
for i=1:1:nb
   Matfiles{i} * Matfiles{i};
end

% NETSOLVE (non-blocking calls for parallelism)
for i=1:1:nb
   request(i) = netsolve_nb('send,''square,
   'Matfiles{i},Matfiles{i});
end

... do some work locally...

for i=1:nb
   square{i} = netsolve_nb('wait,'request(i));
end
```

Figure 3. Matlab script sketches.

Integrating parallel numerical libraries

Integrating parallel packages into NetSolve will let users on workstations access MPP systems to perform large computation. This access can be extremely simple and the users might not even be aware that they are using a parallel library. Furthermore, this parallel library will be accessible for C, Fortran, Matlab, Java programs, and even a Java GUI.

NetSolve views a computational resource as a NetSolve server running on some platform. The specifics of that platform are totally hidden from the user. Therefore, a user could still use the simple NetSolve interfaces to access the power of MPPs or networks of workstations to perform large computations.

Integrating parallel software packages into NetSolve

ScaLapack (Scalable Linear Algebra Package) is a library of high-performance linear algebra routines for distributed-memory message-passing multiple-instruction, multiple-data computers as well as networks of workstations supporting PVM[15] or MPI.[16] Developed at the University of Tennessee, Knoxville, the Oak Ridge National Laboratory, and the University of California, Berkeley, ScaLapack is a continuation of the Lapack project,[17] which designed and produced analogous software for workstations, vector supercomputers, and shared-memory parallel computers. The ScaLapack library contains routines for solving systems of linear equations, least-squares problems, and eigenvalue problems. ScaLapack views the underlying multiprocessor system as a rectangular process grid. Global data maps to the local memories of the processes in that grid assuming a 2D block-cyclic distribution of dense matrices. See the latest edition of the *ScaLapack User's Guide* for further details.[18]

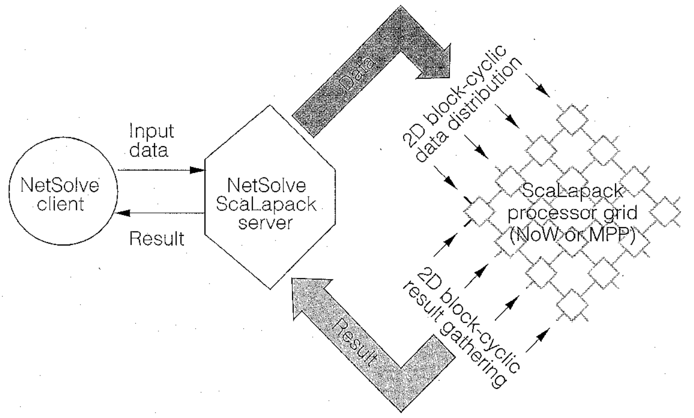
Input data
NetSolve client
NetSolve ScaLapack server
Result
Data
2D block-cyclic data distribution
ScaLapack processor grid (NoW or MPP)
2D block-cyclic result gathering
Result

Figure 4. The ScaLapack NetSolve server paradigm.

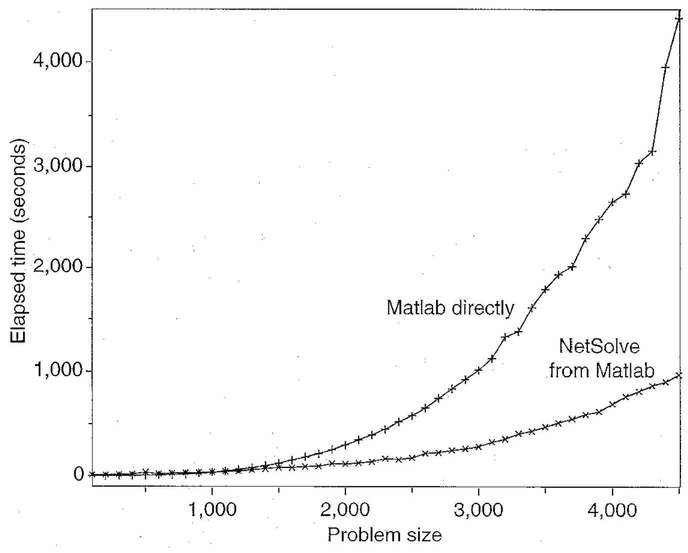

Figure 5. Using ScaLapack transparently from Matlab via NetSolve.

Figure 4 is a sketch of the ScaLapack NetSolve server. The server receives data input from the client in the traditional way. After setting up the processor grid, the server distributes the input data in a 2D block-cyclic fashion and initiates the call to ScaLapack. When the ScaLapack call returns, the result of the computation is 2D block-cyclic distributed as well. The server then gathers that result and returns it to the client in the traditional way. This process is completely transparent to the user who does not even realize that a parallel execution is taking place.

This approach is very promising. A first prototype of the ScaLapack NetSolve server is available in NetSolve version 1.1.b. Figure 5 shows the results of an experiment where a Matlab user can possibly contact a ScaLapack NetSolve server running on a cluster of eight processors that are identical to the processor running Matlab. The graph shows the elapsed times using Matlab or NetSolve for solving a linear system with increasing matrix sizes. As expected, for large matrices, the use of ScaLapack leads to much better performance. Users only need to make a small syntactic change to their input to obtain such performance. Namely, to use Matlab directly, users should type in

```
[x] = a\b;
```

and to use NetSolve from Matlab

```
[x] = netsolve('linsol,'a,b);
```

There are many research issues arising with integrating parallel libraries in NetSolve. First, the agent needs to perform performance predictions for parallel algorithms running in various distributed systems. Such predictions will be much more involved than those the agent currently performs. Furthermore, the agent might have to make choices regarding, for example, the use of ScaLapack or Lapack for a given problem. Answering the question "is it better to send this particular computation to a sequential Lapack server or a parallel ScaLapack server?" will certainly be difficult in many cases. Second, the server itself must make choices concerning the processor grid size and the block size of the 2D block-cyclic distribution. This choice usually depends on the nature and size of the computation to be performed. In the current prototype, we did not pay much attention to such choices, but a more realistic version must yield the best performance for the number of processors available. Other issues are related to the actual operating system of the hardware platform and include processor availability and batch systems.

NetSolve and Ninf

Ninf, developed at the Electrotechnical Laboratory, Tsukuba, allows users to access computational resources including hardware, software, and scientific data distributed across a wide-area network.[6] To facilitate location transparency and network-wide parallelism, the Ninf MetaServer maintains global resource information regarding computational server and databases. It can therefore allocate and schedule coarse-grained computations to achieve good global load balancing. Clearly, NetSolve and Ninf bear strong similarities both in motivation and general design.

A gateway between Ninf and NetSolve

A collaboration between the two projects seems natural. However, the Ninf MetaServer and the NetSolve Agent, even though similar in intent, have different philosophies. The NetSolve Agent is really a resource broker, whereas the Ninf MetaServer is a proxy. This, along with many other less fundamental issues, prevents a seamless integration of the two systems.

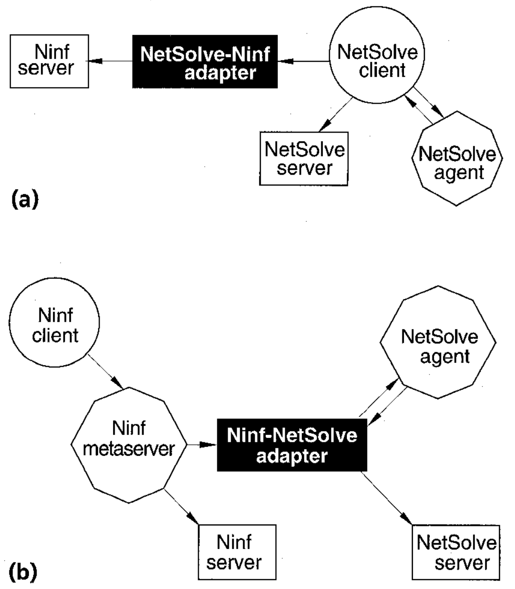

Figure 6. Gateway: (a) going from NetSolve to Ninf; (b) going from Ninf to NetSolve.

To overcome these issues, the Ninf team started developing an adapter so that Ninf clients can use NetSolve resources and vice versa. This first implementation of the adapter is written in Java, and is depicted in Figure 6.

Figure 6a shows the Ninf-NetSolve adapter allowing access to Ninf resource from a NetSolve client. The adapter is just seen by the NetSolve agent as any other NetSolve server. The adapter performs protocol translation, interface translation, and data transfer that is transparent to the NetSolve client. The Ninf server performs a computation and returns the result to the user.

In Figure 6b, the NetSolve-Ninf adapter is seen by the Ninf MetaServer as a Ninf server, but in fact plays the role of a NetSolve client. This is a little different from the Ninf-NetSolve adapter because, as we've said, the NetSolve agent is a resource broker, whereas the Ninf MetaServer is a proxy server. Once the adapter receives the result of the computation from some NetSolve server, it transfers that result back to the Ninf MetaServer that forwards it to the Ninf client.

Evaluations of the first implementation show that using either system via an adapter causes acceptable overheads, mainly due to additional data transfers.

Using NetSolve to extend ImageVision

NetSolve can serve as a building block for a general-purpose framework for basic image processing.

Integrating the ImageVision Library into NetSolve

A project under development at the ICG Institute at Graz University of Technology, Austria, focuses on making basic image-processing functions available for remote execution over a network. This project includes two objectives that can be leveraged by NetSolve. First, the resulting software should prevent the user from having to install complicated image-processing libraries. Second, the functionalities should be available via Java-based applications. The ImageVision Library (IL)[19] developed by Silicon Graphics contains typical image-processing routines to access, manipulate, display, and store image data. Because it is multithreaded, it can make use of multiprocessor machines and other special graphic hardware. Because the ICG research team judges that ImageVision is quite complete and mature, they think it is a good choice for an engine for building a remote-access image-processing framework. Such a framework will make IL accessible from any platform (and not only from SGI workstations).[20]

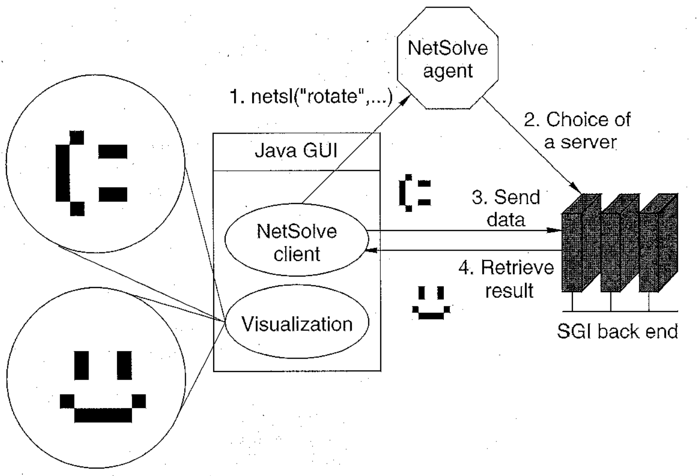

Figure 7. ImageVision and NetSolve.

NetSolve as an operating environment for ImageVision

The reasons for making NetSolve the first choice for such a project are diverse:

- NetSolve is easy to understand, extend, and use at the client level.
- NetSolve is freely available.
- NetSolve provides language binding to Fortran, C, and Java.
- NetSolve's agent-based design allows load balancing among the available servers.

The researchers at ICG have created new NetSolve computational modules corresponding to the desired image-processing functionalities, and the servers grant access to these functionalities via all the NetSolve user interfaces. They have built a Java GUI to IL on top of the NetSolve Java API.

Figure 7 shows a simple example of how ImageVision can be accessed via NetSolve. As shown, the Java GUI offers visualization capabilities. For computations, it uses an embedded NetSolve client and contacts SGI servers that have access to IL. The protocol depicted is simplistic for now. To obtain acceptable levels of performance, we must minimize the network traffic. There are several ways of approaching this problem. We could keep a *state* in the server, meaning that some server always keeps the most recent image objects in memory. We could also pack several operations in one single request. We are currently investigating these possibilities.

We have discussed throughout this article how NetSolve can be customized, extended, and used for a variety of purposes. Perhaps the most important issue we raised concerns the agent resource-management strategy. The NetSolve agent performs performance predictions for all the suitable resources when it receives a user request. Such predictions are much more difficult to realize for a Condor pool or a parallel machine than for a single workstation. Much work will be devoted to unifying our performance metrics so that they encompass the specifics of these new types of resources. We have also shown how network-enabled servers, and NetSolve in particular, have helped in developing a more general image-processing framework. All these developments take place at different levels in the NetSolve project and have had and will continue to have an impact on the project itself, causing it to improve and expand.

NetSolve is an environment for networked computing whose goal is to deliver the power of computational grid environments to users who need processing power but are not expert computer scientists. It achieves this goal with its three-part client-agent-server architecture. In order to deliver the full capacity of grid resources, NetSolve must deal with the potential for these resources to be unstable, which means that fault-tolerance and/or computation migration must be employed. We have described how the current version of NetSolve addresses these issues, and how NetSolve will evolve to address them more completely. ♦

References

1. *Matlab Reference Guide*, The Math Works, Natick, Mass., 1992.
2. H. Casanova, J. Dongarra, and K. Seymour, *Client User's Guide to NetSolve*, Tech. Report CS-96-343, Dept. of Computer Science, Univ. of Tennessee, Knoxville, 1996.
3. H. Casanova and J. Dongarra, "NetSolve: A Net-

work Server for Solving Computational Science Problems," *Proc. Supercomputing '96*, IEEE Computer Society Press, Los Alamitos, Calif., 1996.

4. J.R. Stiles et al., "Miniature Endplate Current Rise Times ¡100 μs from Improved Dual Recordings Can Be Modeled with Passive Acetylcholine Diffusion from a Synaptic Vesicle," *Proc. Nat'l Academy Science,* National Academy of Science, Washington, D.C., Vol. 93, 1996, pp. 5747–5752.
5. V. Menon and A. Trefethen, "MultiMatlab: Integrating Matlab with High-Performance Parallel Computing," *Proc. Supercomputing '97*, IEEE CS Press, 1997.
6. S. Sekiguchi et al., "Ninf: Network based Information Library for Globally High Performance Computing," *Proc. of Parallel Object-Oriented Methods and Applications (POOMA)*, 1996.
7. J. Czyzyk, M. Mesnier, and J. Moré, *NEOS: The Network-Enabled Optimization System*, Tech. Report MCS-P65-1069, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1996.
8. I. Foster and K. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," to be published in *Proc. Workshop on Environments and Tools*, SIAM, Philadelphia, 1998.
9. A. Grimshaw et al., *A Synopsis of the Legion Project,* Tech. Report CS-94-20, Dept. of Computer Science, Univ. of Virginia, Charlottesville, Va., 1994.
10. T.A. Howes, *The Lightweight Directory Access Protocol: X.500 Lite,* Tech. Report CITI-95-8, CITI, Univ. of Michigan, Ann Arbor, Mich., 1995.
11. K. Moore et al., *Resource Cataloging and Distribution System,* Tech. Report CS-97-346, Computer Science Dept., Univ. of Tennessee, 1997.
12. I. Foster et al., "A Fault Detection Service for Wide Area Distributed Computations," to be published in *Proc. Symp. High Performance Distributed Computing, IEEE CS Press.*
13. A. Pope, *The CORBA Reference Guide,* Addison-Wesley, New York, 1998.
14. T.B. Downing, *RMI—Remote Method Invocation,* IDG Books Worldwide, Foster City, Calif., 1998.
15. A. Geist et al., *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing,* MIT Press, Cambridge, Mass., 1994.
16. M. Snir et al., *MPI: The Complete Reference,* MIT Press, 1996.
17. E. Anderson et al., *Lapack Users' Guide, Second Ed.,* SIAM, 1995.
18. L.S. Blackford et al., *ScaLapack Users' Guide,* SIAM, 1997.
19. G. Eckel, J. Neider, and E. Bassler, *ImageVision Library Programming Guide,* Silicon Graphics, Mountain View, Calif., 1996.
20. M. Oberhuber, "Integrating ImageVision into NetSolve," 1997; http://www.icg.tu-graz.ac.at/mober/pub.

Henri Casanova is a postdoctoral research associate in the University of Tennessee's Computer Science Department. His research interests include all areas of metacomputing, theoretical models for predicting and forecasting the performance of globally or locally distributed applications, stochastic methods, statistical inference, sampling theory, large-deviation theory, and time-series analysis. He received his BS in computer science and applied mathematics from the Ecole Nationale Supérieure d'Electrotechnique, d'Infomatique et d'Hydraulique de Toulouse (ENSEEIHT), his MS in parallel architectures and applied mathematics from the University Paul Sabatier, Toulouse, and his PhD in computer science from the University of Tennessee, Knoxville. Contact him at 104 Ayres Hall, Dept. of Computer Science, Univ. of Tennessee, Knoxville, Tenn., 37996; casanova@cs.utk.edu; http://www.cs.utk.edu/~casanova/.

Jack Dongarra holds a joint appointment with the University of Tennessee and Oak Ridge National Laboratory under the UT/ORNL Science Alliance Program. He specializes in numerical algorithms, parallel computing, use of advanced computers, programming methodology, and tools for parallel computers, as well as the development, testing, and documentation of high-quality mathematical software. He was involved in the design and implementation of the software packages Eispack, Linpack, the BLAS, Lapack, ScaLapack, Netlib, PVM, MPI, and the National High-Performance Software Exchange. He received his PhD in applied mathematics from the University of New Mexico. He is co-editor-in-chief of the *International Journal of Supercomputer Applications, Netlib,* and the *SIAM Series on Software, Environments, and Tools for Scientific Computing.* Contact him at 104 Ayres Hall, Dept. of Computer Science, Univ. of Tennessee, Knoxville, Tenn., 37996; dongarra@cs.utk.edu; http://www.netlib.org/utk/people/JackDongarra/.