Using Agent-based Software for Scientific Computing in the NetSolve System

Henri Casanova* Jack Dongarra* †

August 9, 1997

Abstract

Agent-based computing is increasingly regarded as an elegant and efficient way of providing access to computational resources. Several metacomputing research projects are using *intelligent agents* to manage a resource space and to map user computation to these resources in an optimal fashion. Such a project is NetSolve, developed at the University of Tennessee and Oak Ridge National Laboratory. NetSolve provides the user with a variety of interfaces that afford direct access to preinstalled, freely available numerical libraries. These libraries are embedded in computational servers. New numerical functionalities can be integrated easily into the servers by a specific framework. The NetSolve agent manages the coherency of the computational servers. It also uses predictions about the network and processor performances to assign user requests to the most suitable servers. This article reviews some of the basic concepts in agent-based design, discusses the NetSolve project and how its agent enhances flexibility and performance, and provides examples of other research efforts. Also discussed are future directions in agent-based computing in general and in NetSolve in particular.

Keywords
Agent, Metacomputing, Client-Server, Scientific Computing, Networking, Load Balancing, Fault Tolerance, Computational Servers

*Department of Computer Science, University of Tennessee, TN 37996
†Mathematical Science Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831

1 Introduction

Scientific computing has applications in many fields of science and engineering and hence has become a major focus of research centers and industry. The demand for scientific problem-solving facilities has led to the development of numerous software tools usable on diverse hardware platforms. Some numerical tools, such as MATLAB[1] or Mathematica [2], have enjoyed great success. They generally provide an interactive interface, a set of built-in functionalities, and the possibility of writing scripts to perform complex computations.

Another class of tools falls under the category of numerical function libraries. These libraries are less convenient than interactive tools because the user is required to write an actual program (generally in C or Fortran). Nevertheless, they offer several advantages. A large number of such libraries exists, and they cover diverse fields of computational science. Moreover, unlike most interactive tools, numerical libraries are often freely available and can be downloaded directly from the World Wide Web.

A third class of tools comprises runtime packages whose goal is to help the user perform some specific types of scientific computations. Like numerical libraries, these packages are usually freely available. One example is the NEOS project [3], which is focused on linear programming and optimization. However, most such tools are not yet well established, and the user often feel the need for numerical capabilities that are outside their scope.

Users wanting to solve a numerical problem are thus confronted with a dilemma. They can purchase an expensive commercial product and take the risk that it might not be suitable for future use on different kinds of problems, or they can try to locate and download free libraries and write programs in terms of specific functions or subroutines. In this article, we focus on the second situation.

The user who decides to use free software libraries must first look for the appropriate library or set of libraries needed for his specific computational problem. Usually, such libraries can be found in established software repositories. A well-known repository, for example, is Netlib [4], which is maintained through the collaborative effort of several institutions and universities. Software repositories present some intrinsic difficulties for the unexperienced user. First, they are usually very large. For example, Netlib contains more than 40 different numerical libraries, a number that amounts to more than 20,000 different numerical subroutines or functions. And Netlib is just one of many such repositories. The National HPCC Software Exchange (NHSE) [5] Web site gives access to more than 50 different HPCC-related repositories. Second, software repositories contain very different types of libraries. Having such diversity can be regarded as a strength of the whole software repository approach: it guarantees that the repositories will target a broad scope of users. However, because of this diversity, each time a user downloads a new library, he must go through a new learning phase.

Once the appropriate library has been located, it must be downloaded and installed. Depending on the nature of the software, this step might be nontrivial. For instance, a lot of freely distributed software is not fully portable: it is not uncommon to have comments in the code of the numerical software saying that such and such sections will run only on certain systems. Another example is software using system libraries that might not be directly available on the user's system. To an experienced user, these problems are not really threatening. They can generally be solved by a sequence of compilations, linkings, and system libraries searching. However, all these steps can take a large amount of time if the user has not much experience in software installation. The biggest steps still remain—learning how to use the library itself and learning how to write a program in terms of its component. These tasks can be formidable and time-consuming (without even mentioning the debugging phase).

Clearly, much work remains in order to enable a straightforward use of freely available numerical function libraries. Instead of leaving the user responsible for exploring large software repositories and installing the downloaded software, an alternative approach is to provide the user with an infrastructure that facilitates access to numerical libraries through a variety of interfaces. Such an infrastructure could also manage the different hardware and software computing resources available to the user, with a view to maximizing performance. In this article, we describe an agent-based environment that provides these capabilities. The environment, called *NetSolve*, is being developed as part of a project at the University of Tennessee and Oak Ridge National Laboratory.

The article is organized as follows. Section 2 gives an overview of NetSolve. Section 3 develops the agent concept and relates it to several ongoing research works. Section 4 describes in more detail the way the NetSolve agent operates; examples are provided to illustrate the benefits of the agent-based design. Section 5 discusses future directions in the NetSolve system, and Section 6 draws some conclusions.

2 Overview of NetSolve

In this section we describe basic concepts that are fundamental for agent-based network-enabled software. The concepts are relevant not only to NetSolve but to other projects with similar goals.

2.1 Basics

NetSolve is a client-server network-based system. One can distinguish three main paradigms for such 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, for example. 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 then is sent back to the user's machine. NetSolve uses this third paradigm.

Figure 2.1 depicts the basic layout of the system. NetSolve provides the user with a pool of computational servers that have access to ready-to-use numerical software. As shown in the figure, the computational servers can be running on single workstations, networks of workstations that can collaborate for solving a problem, or massively parallel systems. The user is using one of the NetSolve client interfaces. Such interfaces allow a user to send requests to the NetSolve system asking for numerical computation to be carried out by one of the servers. The main role of the NetSolve agent is to process this request and to choose the most suitable server for this particular computation. The basic concepts of such an agent are described in Section 3.

One of the major advantages of this approach is that the agent performs load balancing among the different resources. Once a server has been chosen, it is assigned the computation, uses its available numerical software, and eventually returns the results to the user.

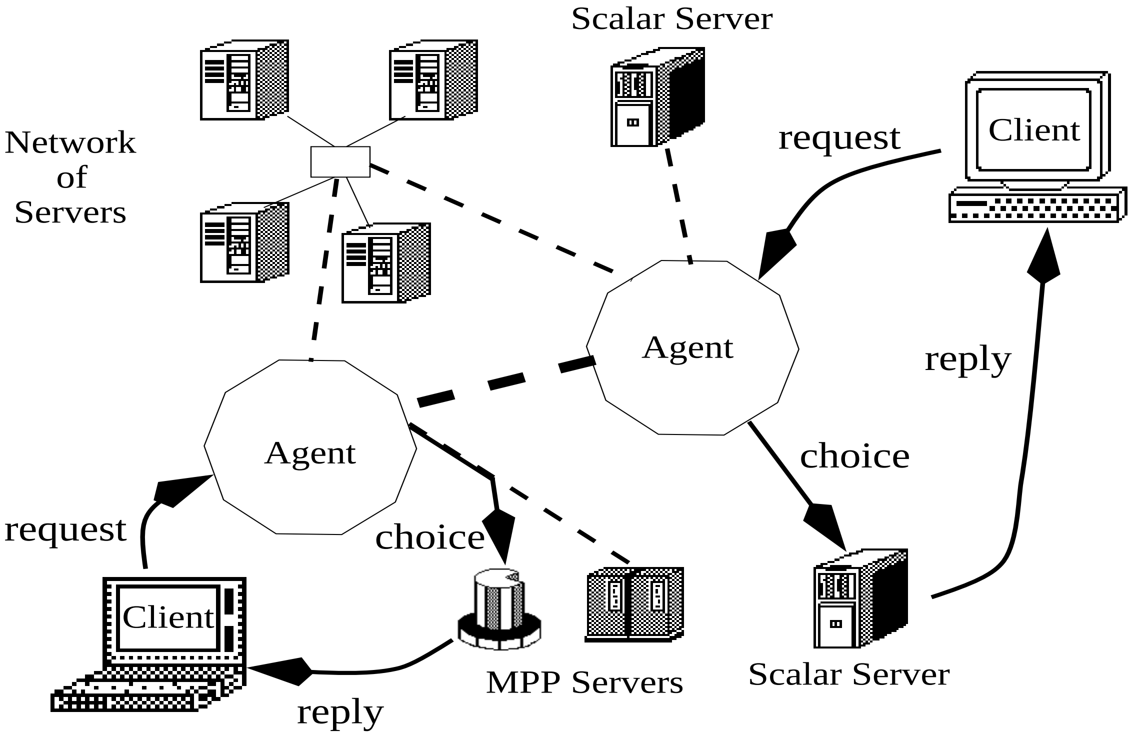

Figure 1: NetSolve's organization

As shown in Figure 2.1, 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 informations about their different servers and allow access from any client to any server, if desirable. Suppose, for example, the set of computational resources spans several local area networks and that users on each of these networks want to use NetSolve to perform scientific computations. It is possible to start a NetSolve agent on each network, so that user requests always go to the "closest" agent to be processed. Different instances of the NetSolve agent can then have different views of the set of computational resources, reflecting the fact that certain clients are closer to certain computational resources. NetSolve can be used via the Internet or on an intranet, inside a research department or a university, without participating in any Internet computation.

Another important aspect of NetSolve is that the configuration of the system is entirely flexible. Any server/agent can be stopped and (re)started at any time without jeopardizing the integrity of the system.

In addition to being a resource broker for the clients, the NetSolve agent is also the primary participant in the management of the different computational resources (hardware and software) and is also in charge of the fault-tolerance mechanisms. Details on the way the NetSolve agent operates and its various responsibilities in the system are given in Section 4.

2.2 The Computational Resources

2.2.1 Challenges

One of the challenges in building the NetSolve system was to design a suitable model for the computational servers. Indeed, when designing such servers, we were faced with the very same difficulties as those highlighted in Section 1. For the user to be able to use numerical software directly through our servers, three major features seemed to emerge as mandatory:

Uniform access to the software: The servers should give the illusion that users have access to a uniform set of subroutines or functions. That is, we wish to hide the specificities of the underlying numerical softwares as much as possible. This feature will ensure that users need not go through 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 add functionalities to a computational server in an easy way. This would give our system the ability to extend and encompass new numerical applications at will.

Preinstallation: The user should not be responsible for installing any numerical software. The software present on the servers should be ready to use, already compiled to the target architecture. Or, in a more general view, the system could dynamically take care of installation and compilation itself, without any intervention from the user.

2.2.2 The Solution

To make the implementation of such a computational server model possible, we have designed a machine-independent, general way of describing a numerical computation, as well as 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 of a computational server. Files written in this language can be compiled by NetSolve into actual computational modules executable on any UNIX platform.

This approach has several advantages. Machine independence is, of course, one advantage, as is the ability of integrating any arbitrary software 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. These files can then be exchanged by any institutions wanting to set up servers, compiled and run to create a new stand-alone NetSolve system or to contribute new servers to an existing system. Each time a new description file is created, the capabilities of the entire NetSolve system are thereby increased.

These advantages, however, are effective only if the process of creating new input files and adding them to a computational server is reasonably straightforward. For this reason, we developed a graphical interface to handle the generation of the description files. The interface performs error checking of the user's input. Using the interface is much easier than creating a description file manually, especially as the complexity of the problem to be described increases.

Not only is this interface graphical, but it is also written in Java. Several factors motivated this choice. First, Java allows one to write GUIs (graphical user interfaces) very easily, thanks to its built-in widget classes. Second, Java is object oriented and therefore provides a good degree of modularity and data encapsulation. These features are important because we might have to modify the syntax of the language in the future to describe wider classes of numerical computations. Third, Java is Web-enabled. This interface could thus be downloaded as an applet, and users setting up NetSolve computational servers could create their description files directly from within Web browsers. These files could then be downloaded from the Web browser and compiled into NetSolve computational modules (thanks to the compiler provided by the NetSolve server software).

The ultimate goal would be to have a NetSolve description file repository on the Web. From that repository, description files could be downloaded at will to set up computational servers. The actual numerical software should also be available to make the creation of these servers almost immediate. The idea would then be to add a complementary repository containing NetSolve description files to a regular software repository (like Netlib [4]).

2.2.3 Existing Resources

So far, description files have been written for the following numerical libraries: FitPack [6], ItPack [7], MinPack [8], FFTPACK [9], LAPACK [10], BLAS [11, 12, 13], QMR [14], and ScaLAPACK [15].

NetSolve computational servers providing access to these packages are running on a 24-hour basis at the University of Tennessee and at other locations world-wide. Real-time information on the available servers and numerical softwares can be found on the NetSolve homepage located at `http://www.cs.utk.edu/netsolve`.

These numerical libraries cover several fields of computational science, including linear algebra, optimization, and fast Fourier transforms. Two particular numerical libraries require special treatment by NetSolve. In the MINPACK (a package of optimization/minimization software) library, for example, the user must supply a piece of code that implements the function to be minimized. The current version of the NetSolve software handles user-supplied functions in a way described in [16]. The ScaLAPACK library is also a special case because it is a parallel library. A prototype description file for ScaLAPACK has been written and tested on a network of workstations managed by PVM [17]. Integrating ScaLAPACK in NetSolve will allow a user on a workstation to access massively parallel systems in order to perform large computations. As outlined in Section 2.3 and detailed in [16], this access is extremely simple, and users will not 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 (as explained in the following section).

2.3 The Client Interfaces

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 C or Fortran program, or from interaction with a high-level interface.

All the interfaces are described in detail in the "NetSolve's Client User's Guide" [16]. In Section 4, we show an example of NetSolve that takes advantage of our agent-based strategy.

3 What Is an Agent?

The fundamental conception behind NetSolve is that *agent-based scheduling can provide robust and high-performance application execution in a metacomputing environment.* An agent is an entity (process) that reasons about and makes decisions about the mapping of computations to resources. Consequently, an agent must be able to access application- and system-specific information.

Application-specific information can be, for example, the number and type of the application data structures, or the complexity of the algorithm to be used, or the frequency and volume of communications and computations for a parallel application, or the amount of memory required by the application. *System-specific information,* on the other hand, involves resource performance: number of computational resources available, network characteristics, and, for each of these resources, the amounts of memory available, CPU maximum speeds, and the like.

The agent also needs *dynamic information* to determine the system state. Before mapping computations on resources, the agent must take into account whether the candidate CPUs are lightly or heavily loaded, whether the network itself is loaded, and what the best combination of resources is at any given time. Combining the application-specific and system-specific information with dynamic information describing system conditions allows the agent to assess the cost of a given mapping, and thus to choose the optimal one with respect to the user's performance criteria. Rather than controlling the resources in a way that maximizes application performance, an agent *controls the mapping of the computations* so that it makes maximal use of the available resources.

This general concept is being used by several metacomputing research projects and is widely extended and adapted to fit the requirements and specifications of these projects. Here we discuss two such projects and distinguish their features from those of NetSolve. In the next section, we describe the NetSolve agent.

The *AppLeS* (application-level schedulers) system [18], developed at the University of California - San Diego, focuses on scheduling agents for parallel *metacomputing* applications. In the AppLeS framework, the agent performs the mapping of computation to resources in the form of a particular scheduling of the user's parallel application. To determine schedules, the agent must consider the requirements of the application and the predicted load and availability of the system resources at scheduling time. Each AppLeS agent has access to dynamic predictions of resource performance, thanks to the NWS (Network Weather Service)[19] that has been developed as a separate facility. AppLeS and NetSolve differ in that AppLeS is concerned with the scheduling of stand-alone parallel applications whereas NetSolve schedules computational requests but provides access to numerical software.

The Network-based Information Library (Ninf) [20] project developed at the Electrotechnical Laboratory in Tsukuba is using an agent-based design. The agent in Ninf, called a *meta-server*, is responsible for the request assignment to the different servers. Ninf and NetSolve have several common aspects even though NetSolve seems to be a little more mature at this time. This common ground has motivated a collaboration between the University of Tennessee and the Electrotechnical Laboratory to built a common interface between the two systems. A prototype of a Ninf-NetSolve *adapter* has already been developed. The protocols of both projects will evolve to permit a seamless integration of both systems. Eventually, Ninf and NetSolve users will be able to use any interface of any project to perform computations on servers administrated either by Ninf or by NetSolve.

4 The NetSolve Agent

In this section, we highlight the main responsibilities of the agent in the NetSolve system, and we give some details about its current implementation.

4.1 The Agent as a Database

Keeping track of what software resources are available and on which servers they are located is perhaps the most fundamental responsibility of the NetSolve agent. Since the computational servers use the same framework to contribute software to the system (see Section 2.2.2), the agent can maintain data describing the different numerical functionalities available to the users.

The protocol is fairly straightforward. Each time a new server is started, it sends a registration request to an instance of the NetSolve agent. This request contains general information about the server (including its location), but also the list of numerical functions it intends to contribute to the system. The agent examines this list for possible discrepancies with the other existing servers in the system. Based on the agent's verdict, the server is either rejected or integrated into the system.

Once a new server is accepted, it can be used by a client. The next section explains how the agent chooses a server.

4.2 The Agent as a Resource Broker

The goal of the NetSolve agent is to choose the best-suited computational server for each incoming request to the system. To do so, 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 (e.g., size of the matrices for a linear algebra computation), and so on. Resource-specific information, as explained in Section 3, is composed of static and dynamic data. Static system-specific data is communicated to the agent by each server when it is first started and 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 is the load of the server's host and the network delays and transmission rates to contact that host. The network performance is estimated by the agent by constantly averaging samples of the network delays between the hosts. The strategy for the load of the servers is different: the computational servers notify the agent of their workload fluctuations when they deem it necessary. Rationale and further detail on these protocols can be found in [21].

4.3 Fault Tolerance

As previously mentioned, the hosts in the NetSolve system can be located anywhere on the Internet and can therefore be administered by different institutions. Hence, NetSolve does not try to impose any control on the different resources. This approach is, of course, very flexible, but it requires 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. Every instance of the agent is a repository of the hardware resource. It is therefore natural that the fault-tolerance mechanisms in NetSolve be at least partly implemented by the agent.

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 is started. If the contacted server fails during the computation, then another server is contacted and the computation is restarted. This whole process is transparent to the user. Each time a computational server malfunction (server unreachable, server stopped, failure during computation) is detected by a client, this client notifies the failure to one agent. The agent updates its *tables* and takes the necessary measures. If all the servers have failed, the user is notified that the computation cannot be performed at that time.

4.4 Simple Example of the Agent's Effectiveness

Several simple experiments can be done with the current version of the NetSolve software in order to measure different performance issues. The experiment we describe here provides information about the typical gain a user can obtain by using NetSolve. In this experiment, the user is using MATLAB on a Sun workstation (Sparc 5) to perform several matrix multiplications. The size of the matrix is 800 by 800, and the use performs from 1 to 16 multiplications. A NetSolve system is available consisting of seven computational servers. These servers also run on Sun workstations (Ultra 1's) and are located at the University of Tennessee. We note here that the results of this experiment would be the same if, instead of one user, several users were sending requests to the NetSolve system.

Figure 4.4 shows the total execution times in in different situation. The first curve is labeled 'MATLAB' and shows the execution times for 1 to 16 matrix multiplications when the user is using MATLAB directly. The total execution time increases linearly with the number of operations performed, with a rate equal to the execution time of one matrix multiplication. This result was expected, since the multiplications are executed one after the other.

The other four curves are all labeled 'NetSolve' and show the execution times when the user is using the MATLAB interface to NetSolve to perform his computations. Each curve correspond to a different location of the user's machine. The 'Intranet' curve shows execution times when the user's machine is located on the network of the Computer Science Department at the University of Tennessee. For the 'Close Internet' curve, the user is at the Oak Ridge National laboratory. The 'Continental Internet' curve shows execution times for a user at the University of California at Berkeley, and the 'Overseas Internet' curve is for a user at the Danish Technical University in Copenhagen, Denmark. For these four curves, the user's machines were all Sun workstations (Sparc 5), and the measurements were taken from 11:00PM till 4:00AM EST during week days.

There are mainly two facts of interest shown on figure 4.4. First, if the client machine is *close* enough to the NetSolve system (Internet-wise), then the NetSolve overhead (contacting the agent, sending the input data, retrieving the result) is still small enough to allow a win over MATLAB computation. This is of course true because the server machines are faster than the client machine. On the curves, this corresponds to the first points (one multiplication) and we can see that NetSolve

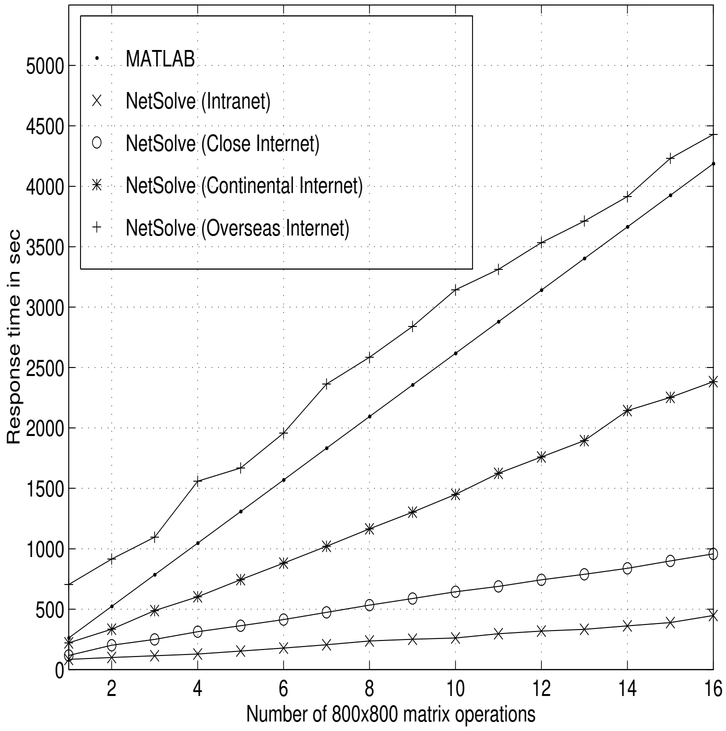

Figure 2: Multiple 400x400 matrix multiplications

wins over MATLAB for all but the overseas user. In fact, there is a very large performance gap between the overseas Internet curve and the other ones due to very poor network bandwidth.

Second, the execution times when using NetSolve are sensibly linear. The rates are approximatively 22 for the intranet client, 52 for close Internet, 135 for distant Internet, 232 for overseas Internet and 261 for MATLAB. The NetSolve rates are all lower than the MATLAB rate, and of a different order of magnitude when the user is *close* enough to the NetSolve system. This is due to the load-balancing strategy implemented by the NetSolve agent. All the user requests are processed in parallel on different servers, leading to an impressive speedup. One may be surprised that the execution time is linear even for more that seven multiplications since only seven servers are available. In fact, due to the matrix size, the speed of the processors and the network contentions, the NetSolve overhead is sufficient to prevent a server from performing two multiplications simultaneously. For bigger matrix sizes or faster networks, the execution time would not be linear, but would start increasing faster than its initial rate after seven multiplications.

The NetSolve agent attempts to locate and schedule the user's computation on powerful machines, in order to optimize performance. In this experiment, however, all the servers were identical, as was their workload. The agent was therefore compelled to schedule the computation in what appears to be a round-robin fashion.

For the NetSolve approach to be most worthwhile, users must have to change as little of their code as possible. The explanation is as follows. The MATLAB code to perform a matrix multiply is given below:

```
c = a * b
```

An equivalent NetSolve code using a blocking call could be

```
c = netsolve('matmul',a,b)
```

However, to achieve the speedups in the experiment, the user must call NetSolve in an asynchronous way as

```
request = netsolve_nb('matmul',a,b)
...
...
c = netsolve('wait',request)
```

The price to pay in code complexity is quite reasonable, given the improvement in speed that can be achieved with NetSolve. This is even more true about the C and Fortran interfaces. Even more striking, the same performance can be achieved from a Java program calling the NetSolve Java API or from the Java GUI by simply clicking and pointing in graphical windows.

5 Future Directions

Agent-based computing seems to be a promising strategy. NetSolve will certainly evolve into a more elaborate system in the future, and a major part of this evolution is bound to take place within the agent. We highlight here some of the expected changes in the agent concept.

As the number of users and resources increases, it will be increasingly difficult to maintain a coherent resource space. The issue of a robust and flexible naming strategy will undoubtedly arise. Several naming services have been designed (LDAP [22], RCDS [23]), and implementations are starting to become available. Such services would provide a good basis for a metacomputing project like NetSolve.

NetSolve will eventually need to provide a user-accounting feature so that realistic bounds can be imposed on resource usage. We could, for instance, restrict the access to the resources, restrict the access for some users, or do a combination of the two. The word *restrict* is still to be precisely defined in this context. A practical and convenient scheme would be to use tokens or credits that users can release or spend to perform computations. Different users could have a different level of access. For instance, students would not be permitted to run large computations, whereas researchers could have full access. It also might be advisable to let administrative authorities customize their own accounting policy and put bounds on the usage of their resources. These bounds would be in terms of CPU time or megabytes on hardware resources, or in terms of number of requests. We can imagine, for instance, that two universities or national laboratories could allow each other to use every resources—with, however, a "preference" for the local users to use the local resources. It seems natural to make the NetSolve agent the primary actor of any accounting mechanism.

Finally, as the types of hardware resources and numerical software available on the computational servers become more and more diverse, the resource broker embedded in the agent will need greater sophistication. Some issues have already arisen with the current implementation of NetSolve. For instance, it is difficult for the agent to predict execution times of computations that involve user-supplied functions, or for any iterative algorithm whose complexity depends on the input data. Now that ScaLAPACK [15] is being integrated into NetSolve, predicting performance on multiprocessor configurations must also be explored. Many difficulties arise in providing a uniform performance metric that encompasses any type of algorithmic and hardware considerations in a metacomputing setting. Much work will be devoted to implementing a reasonable approximation of such a metric within the agent.

6 Conclusion

In this article, we have described how agent-based computing can help researchers use freely available numerical software in an easy and efficient way. Several metacomputing research projects have already used an agent-based strategy to manage a heterogeneous pool of resources. These resources can be hardware or software, or a combination of the two.

Featured here is the agent-based project called NetSolve, developed at the University of Tennessee and Oak Ridge National Laboratory. NetSolve provides the user with a pool of computational servers running on single workstations, networks of workstations, or massively parallel systems. These servers also provide uniform access to a variety of numerical software. Moreover, a general framework has been designed so that any arbitrary numerical library can be integrated in a NetSolve computational server. The NetSolve agent schedules user requests and manages the resources. The role of this agent is fundamental, and it is bound to become even more important in the future.

A first version of the NetSolve software, as well as all relevant information about the system, is available at `http://www.cs.utk.edu/netsolve`. We anticipate that the NetSolve software will evolve rapidly and will become more and more interoperable with other metacomputing projects.

References

[1] The Math Works Inc. *MATLAB Reference Guide*. 1992.

[2] S. Wolfram. *The Mathematica Book, Third Edition.* Wolfram Median, Inc. and Cambridge University Press, 1996.

[3] J. Czyzyk, M. Mesnier, and J. Moré. NEOS : The Network-Enabled Optimization System. Technical Report MCS-P615-1096, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[4] S. Browne, J. Dongarra, E. Grosse, and T. Rowan. The Netlib Mathematical Software Repository. *D-Lib Magazine*, Sep. 1995. Accessible at http://www.dlib.org/.

[5] S. Browne, J. Dongarra, S. Green, K. Moore, T. Rowan, R. Wade, G. Fox, K. Hawick, K. Kennedy, J. Pool, R. Stevens, R. Olson, and T. Disz. The National HPCC Software Exchange. *IEEE Computational Science and Engineering*, 2(2):62–69, Summer 1995.

[6] A. Cline. Scalar- and Planar-Valued Curve Fitting Using Splines Under Tension. *Communications of the ACM*, 17:218–220, 1974.

[7] D. Young, D. Kincaid, J. Respess, and R. Grimes. Itpack2c: 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.

[8] J. Moré, B. Garbow, and K. Hillstrom. Minpack : Documentation file accessible at: "http://www.netlib.org/minpack/readme".

[9] P. Swarztrauber. FFTPACK : Documentation file accessible at: "ftp://ftp.ucar.edu/ftp/dsl/lib/fftpack/readme".

[10] 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, Second Edition.* SIAM, Philadelphia, PA, 1995.

[11] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *ACM Transactions on Mathematical Software*, 5:308–325, 1979.

[12] J. Dongarra, J. Du Croz, S Hammarling, and R. Hanson. An Extended Set of Fortran Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–32, 1988.

[13] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[14] R.W. Freund and N.M. Nachtigal. QMR: A quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.

[15] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[16] 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, 1996.

[17] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM : Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing.* The MIT Press Cambridge, Massachusetts, 1994.

[18] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-Level Scheduling on Distributed Heterogeneous Networks. In *Proc. of Supercomputing'96, Pittsburgh.* Department of Computer Science and Engineering 0114, University of California, San Diego, 1996.

[19] R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. Technical Report TR-CS96-494, U.C. San Diego, October 1996.

[20] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima. Ninf : Network based Information Library for Globally High Performance Computing. In *Proc. of Parallel Object-Oriented Methods and Applications (POOMA), Santa Fe*, 1996.

[21] H Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. In *Proc. of Supercomputing'96, Pittsburgh.* Department of Computer Science, University of Tennessee, Knoxville, 1996. to appear in *The International Journal of Supercomputer Applications and High Performance Computing.*

[22] Timothy A. Howes. The Lightweight Directory Access Protocol: X.500 Lite. Technical Report CITI-95-8, CITI, University of Michigan, July 1995.

[23] Keith Moore, Shirley Browne, Jason Cox, and Jon Gettler. The Resource Cataloging and Distribution system. Technical Report UT-CS-97-346, University of Tennessee, 1997.