NetSolve: A Network Server for Solving Computational Science Problems

Henri Casanova* Jack Dongarra*†

November 27, 1995

Abstract

This paper presents a new system, called NetSolve, that allows users to access computational resources, such as hardware and software, distributed across the network. This project has been motivated by the need for an easy-to-use, efficient mechanism for using computational resources remotely. Ease of use is obtained as a result of different interfaces, some of which do not require any programming effort from the user. Good performance is ensured by a load-balancing policy that enables NetSolve to use the computational resource available as efficiently as possible. NetSolve is designed to run on any heterogeneous network and is implemented as a fault-tolerant client-server application.

Keywords

Distributed System, Heterogeneity, Load Balancing,
Client-Server, Fault Tolerance, Linear Algebra, Virtual Library.

University of Tennessee - Technical report No cs-95-313

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

1 Introduction

An ongoing thread of research in scientific computing is the efficient solution of large problems. Various mechanisms have been developed to perform computations across diverse platforms. The most common mechanism involves software libraries. Unfortunately, the use of such libraries presents several difficulties. Some software libraries are highly optimized for only certain platforms, but do not provide a convenient interface to other computer systems. Other libraries may demand considerable programming effort from a user, who may not have the time to learn the required programming techniques. While a limited number of tools have been developed to alleviate these difficulties, such tools themselves are usually available only on a limited number of computer systems. MATLAB (see [1]) is an example of such a tool.

These considerations motivated the establishment of the NetSolve project. NetSolve is a client-server application designed to solve computational science problems over a network. A number of different interfaces have been developed to the NetSolve software, so that users of C, Fortran, MATLAB, or the Web can easily use the NetSolve system. The underlying computational software can be any scientific package, thereby ensuring good performance results. Moreover, NetSolve uses a load-balancing strategy to improve the use of the computational resources available.

This paper introduces the NetSolve system, its architecture and the concepts on which it is based. We then describe how NetSolve can be used to solve complex scientific problems. Our focus is on linear algebra problems, but we emphasize that the NetSolve project easily can be extended in order to solve many classes of problem. For instance, it would be straightforward to use NetSolve for image processing or for PDEs for instance.

2 The NetSolve System

This section presents a short description of the NetSolve system and discusses how users can exploit the system for complex problem solving.

2.1 Architecture

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

The current implementation sees the system as a completely connected graph, without any hierarchical structure. This initial implementation was adopted for simplicity. We expect, however, that in order to manage efficiently a pool of hosts scattered on a large-scale network, future implementations will provide greater structure (e.g., a tree structure) which will limit and group large-range communications.

Figure 1 shows the global conceptual picture of the NetSolve system. In this figure, a NetSolve client send a request to the NetSolve agent. The NetSolve communication servers are the effective implementations of the NetSolve agent for now. The agent choses the "best" NetSolve resource, which is a NetSolve computational server. For more details on this *choice* made by the communication server, see section 4.

Every host in the NetSolve system runs a NetSolve server. We distinguish two types of server: computational and communication servers. A *computational* server is a NetSolve resource as seen in figure 1. A *communication* server is an instance of the NetSolve agent. A good implementation would be to have a NetSolve agent on every local network where several clients are sending request to NetSolve. Of course this is not mandatory and the NetSolve system may contain only one instance of the agent.

An important aspect of this server-based system is that each instance of the agent has its own *view* of the system. Therefore, some agents may be aware of more details than are others, depending on their locations. But eventually, the system reaches a stable state in which every agent possesses all the available information on the system (provided the system does not undergo never-ending modifications).

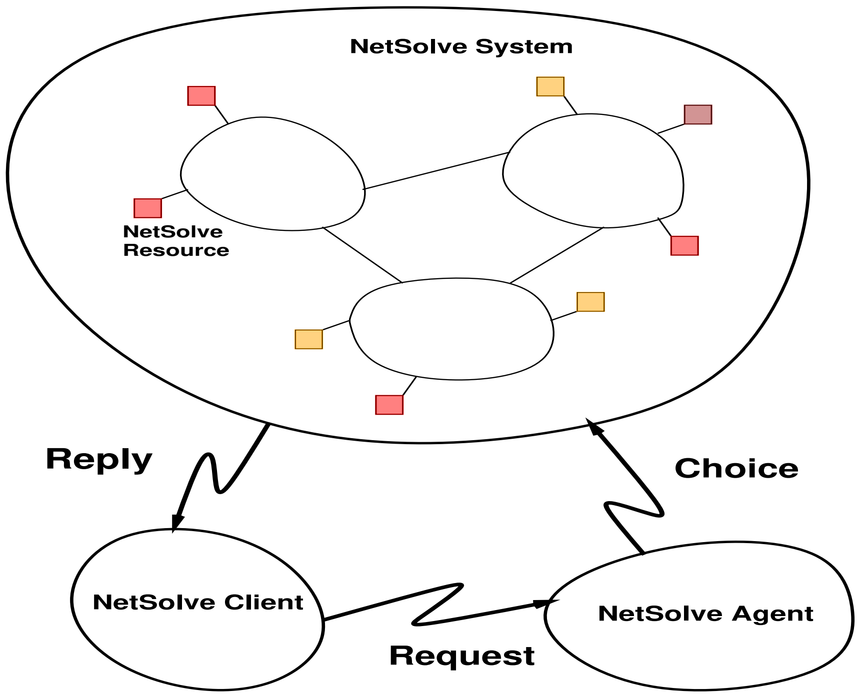

Figure 1: The NetSolve System

From now on, we will use the term *agent*, *agent instance* or *communication server* to design the first process contacted by a client when sending as request.

2.2 NetSolve Management

NetSolve is a fully distributed system and as such needs special features that make its management safe and easy.

Modifying the system. A prerequisite of a distributed system is that it should be able to evolve safely and as easily as possible. To this end, we have designed NetSolve so that new servers can be added at any time. A new server must first contact *any* other server already in the system in order to obtain its own *view* of the system. Then it broadcasts its existence to all the servers of which it is aware, and becomes an independent entity. (Again, the current implementation is completely nonhierarchical.)
In the same vein, it is possible to remove servers from the system at any time, and to restart them if necessary, without modifying the system behavior.

Interactive managing. To manage such a distributed system, we developed an interactive tool that enables the *NetSolve manager* to get information about the system and to modify it dynamically. This tool provides a telnet-like interface through which a user can get data about the configuration, such as the number of participating servers and their characteristics, the number of problems solvable, and the servers on which such problems can be solved.
The following command line, for instance, gives the list of all the servers in the NetSolve system, as it is seen by the server on the host `jupiter`.

```
NetSolve jupiter > list sv
```

```
3 NetSolve servers in the system known by jupiter :
---------------
--> earth (128.145.32.234) (agent)
--> jupiter (132.111.21.134) (resource)
--> saturn (124.122.23.210) (resource)
```

The tool also provides statistics such as the number of problems solved on a given server, the number of requests processed by an instance of the agent or that date at which a server has been started.
Appendix D shows briefly how to add servers to the NetSolve system.

2.3 Protocol Choices

The communication within the NetSolve system is achieved by means of the socket layer. We chose to use the TCP/IP protocol because it ensures a reliable communication between processes. (The fact that a process is limited to a certain number of simultaneous TCP connections was not a problem, given the NetSolve specification.)
To ensure the possible use of an heterogeneous environment, NetSolve uses the XDR protocol between hosts of incompatible data format. Actually, this is the default protocol before two hosts agree that they use the same data format.

3 Interfaces

One of the main goal of NetSolve is to provide the user with the largest number of interfaces and to keep them as simple as possible. We describe here two different kinds of interface: *interactive* and *programming* interfaces.

3.1 Interactive Interfaces

Interactive interfaces offer several advantages. First, they are easy to use because they completely free the user from any code writing. Second, the user still can exploit the power of software libraries. Third, they provide good performance by capitalizing on standards tools like MATLAB. Let us assume, for instance, that MATLAB is installed only on one machine on the local network. It is possible to run NetSolve via MATLAB on this machine and in fact use the computational power of another machine where MATLAB is not available.
The current implementation of NetSolve comprises three interactive interfaces.

3.1.1 The MATLAB Interface

Here is an example of the MATLAB interface to solve an eigenvalues computation :

```
>> A                 = rand(100,100)
>> [real imaginary] = netsolve('DEig',A)
```

This Matlab script first creates a random 100×100 matrix, A. The call to the `netsolve` function returns with the solution in the vectors *real* and *imaginary*, which contains the real parts and imaginary parts of the eigenvalues of matrix A. The call to `netsolve` manages all the NetSolve protocol, and the computation may be executed on a remote host.

3.1.2 The Shell Interface

We also developed a shell interface. Here is the same example as above, with the shell interface :

```
earth % netsolve DEig mat real imaginary
```

Here, `mat`, `real`, and `imaginary` are files. This interface is slightly different from the MATLAB interface because the call to netsolve does not make any difference between inputs and outputs. The difference is made internally, and it is the user's responsibility to know the correct number of parameters. (This number can be obtained from the interactive tool we have developed for NetSolve (see 2.2).

3.1.3 The Graphic Interface

The last of our interactive interfaces is a very attractive one. We developed it with TK/TCL. The aspect of this interface is given in Appendix A. This graphic interface is perhaps the simplest to use. It has been built on top of the shell interface described in the preceding paragraph.

3.2 Programming Interfaces

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

A new feature in these interfaces is that they allow the user to call NetSolve *asynchronously*. By this we mean that it is possible to *post* a request to NetSolve, go on with other computation or other requests, and *poll* for the results later. This asynchronous system has two important reasons. First, it allows the users within their programs to overlap a NetSolve computation by their own computation. Second, it allows the NetSolve system to perform the computation in parallel on different computational servers, by sending several NetSolve requests in a row (see 7).

The NetSolve libraries also provide other functions. For example, they enable the user to contact a NetSolve server in order to obtain some useful informations, as the number of parameters for a given problem.

Simple examples of the C and Fortran interfaces can be found in Appendix B and C.

4 Load Balancing in NetSolve

Load balancing is one of the most attractive features of the NetSolve project. Since NetSolve performs the computations over a network containing a large number of machines with different characteristics, it seems logical to think that one of these machines is the most suitable for a given problem.

Before we consider how NetSolve tries to *guess* which machine is to be chosen, let us examine what criteria determine the *best* machine.

4.1 Calculating the Best Machine

The hypothetical best machine is the one yielding the smallest execution time T for a given problem P. Therefore, we have to compute an estimate of this time on every machine M in the NetSolve system. Basically, we split the time T in T_n and T_c, where

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

The time T_n can be computed knowing the

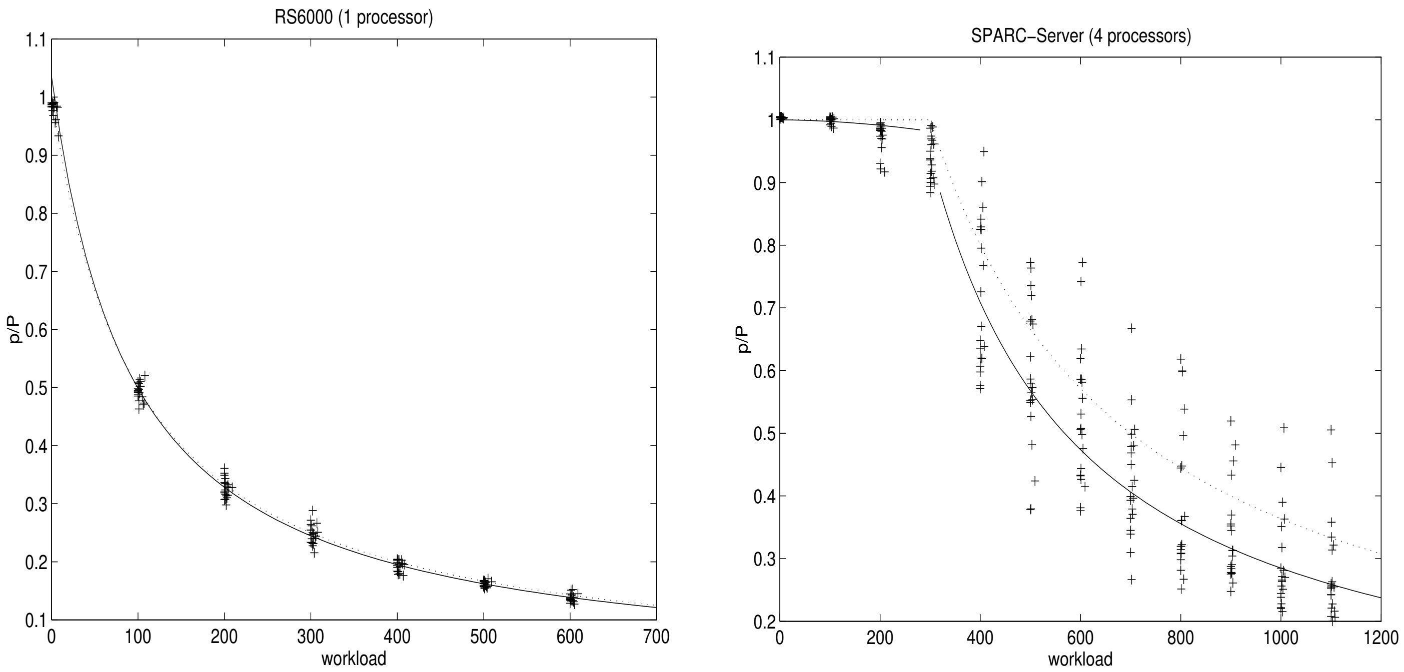

Figure 2: p/P versus workload

1. network latency and bandwidth between the local host and M,
2. size of the data to send, and
3. size of the result to receive.

The computation of T_c involves the knowledge of the

1. size of the problem,
2. complexity of the algorithm to be used, and,
3. performance of M, which depends on
   - the workload of M and
   - the *raw* performance of M.

4.2 Theoretical Model

We have developed a simple theoretical model enabling us to assess the performance, given the raw performance and the workload. This model gives the actual performance, p, as a function of the workload, w; the raw performance, P; and the number of processors on the machine, n:

$$p = \frac{P \times 100 \times n}{100 \times n + max(w - 100 \times (n-1), 0)}$$

To validate this model, we performed several experiments. The results of the experiments are shown in Figure 2, which shows the ratio p/P versus the workload of the machine. Each measure gave one of the "+"

marks. We then computed the mean of all the measures for every value of the workload. An asymptotic interpolation of these mean values is shown with a continuous curve. Our theoretical model is shown with the dashed line.

In Figure 2-(a), we can see that the theoretical model is very close to reality. In Figure 2-(b), since the machine has four processors, the beginning of the curve is a flat line, and the performance begins to drop when the four processors are loaded. Our model is less accurate and always optimistic because it does not take into account any operating system delay to manage the different processors. The *chaotic* behavior of the four-processor machine comes from the fact that the operating system makes some process migrations between the processors.

4.3 Computation of T

The computation of T takes place on a communication server for each problem request and for each computational server M. This computation uses all the parameters listed above. We distinguish three different classes of parameter:

- The client-dependent parameters
  1. The size of the data to send
  2. The size of the result to receive
  3. The size of the problem
- The **static** server-dependent parameters
  1. The network characteristics between the local host and M
  2. The complexity of the algorithm to be used on M
  3. The *raw* performance of M
- The **dynamic** server-dependent parameters
  1. The workload of M

The client-dependent parameters are included in the problem request sent by the client to the communication server. Their evaluation is therefore completely straightforward. The static server-dependent parameters are generally assessed once, when a new server contacts the other NetSolve servers already in the configuration.

Network characteristics. The network characteristics are assessed several times, so that a reasonable average value for the latency and bandwidth can be obtained. We still call them *static* parameters, however, since they are not supposed to changed greatly once their mean value has been computed.

Complexity of the algorithm. When a new computational server joins the NetSolve system, it posts the complexity of all of its problems. This complexity does not change thereafter, since it depends only on the software used by the computational server.

Raw performance. By *raw* performance, we mean the performance of the machine with no other process using the CPU. Its value is determined by each computational server at startup time. We use the LINPACK benchmark to obtain the Kflop/s rate. The LINPACK benchmark computes the "user time" for its run, and therefore corresponds to our definition of raw performance.

4.4 The Workload Information

Workload parameters are the only dynamic server-dependent parameters required to perform the computation of the predicted execution time T. The strategy described above for getting the present workload of every computational server when computing T is absolutely out of the question. Indeed, each problem request would involve a huge number of communications within the NetSolve system, and would therefore be highly inefficient.

Instead, each communication server possesses a cached value of the workload of every computational server. By *cached*, we mean that this value is directly used for T's computation and that it is only updated periodically. Admittedly, this value may be out of date and lead to a wrong estimate of T. Nevertheless, we believe that it is better on the average to take the risk of having a wrong estimate than to pay the cost for getting an accurate one.

We emphasize that we have tried to make this estimate as accurate as possible, while minimizing the cost of its computation. Figure 3 shows the scheme we used to manage the workload broadcast.

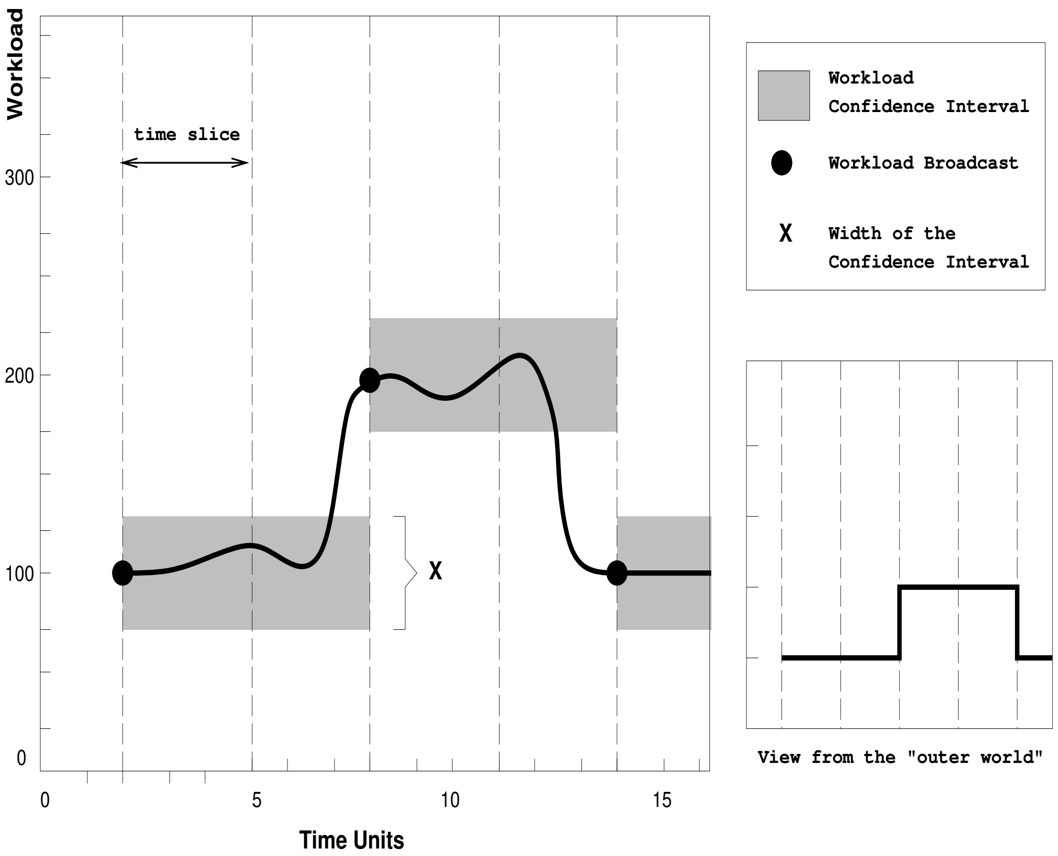

Figure 3: Workload Policy in NetSolve

Let us consider a computational server M and a communication server C. C performs the computation of T according to the last value of M's workload it knows. M broadcasts its workload periodically. In Figure 3, we call *time slice* the delay between to workload broadcast from M. This figure shows the workload function of M versus the time. The simplest solution would be to broadcast the workload at the beginning of each time slice. But experience proves that the workload of a machine can stay the same for a very long time. Therefore, most of the time, the same value would be broadcast again and again over the network. To avoid this useless communication, we chose to broadcast the workload only when it has **significantly** changed. In the figure, we see some shaded areas called the *confidence* interval. Basically, each time the value of the workload is broadcast, the NetSolve computational server decides that the next value to be broadcast should be different enough from the last broadcast one—in other words, outside this confidence interval. In the figure, the workload is broadcast three times during the first five time slices.

Two parameters are involved in this workload management: the width of a time slice and the width of the confidence interval. These parameters must be chosen carefully. A time slice that is too narrow causes the workload to be assessed often, which is very costly in term of CPU cycles. We have to remember that a NetSolve server is supposed to run on a host for a long period of time; it is impossible to let it monopolize a lot of CPU time. The width of the confidence level must also be considered carefully. A narrow confidence interval causes a lot of useless workload broadcasting, which is costly in term of network bandwidth.

Choosing an effective time slice and confidence interval serves another function. It helps to make the workload information on the communication servers as accurate as possible, so that the estimated value of T is reasonable.

We emphasize that experimentation is needed to determine the most suitable time slice and confidence intervals.

5 Fault Tolerance

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

Fault tolerance in NetSolve takes place at different levels. Here we will justify some of our implementation choices.

5.1 Failure Detection

Failures may occur at different level of the NetSolve protocols. Generally they are due to a network malfunction, to a server disappearance, or to a server failure. A NetSolve process (i.e., a client, a server, or a utility process created by a server) detects such a failure when trying to establish a TCP connection with a server. The connection might have failed or have reached a timeout before completion. In this case, this NetSolve process reports the error to the *closest* NetSolve server. (A client would report to its default communication server, a utility process to the server that created it.) The server would simply take the failure into account.

One of the prerequisites for NetSolve was that a server can be stopped and restarted safely. Therefore, all the error reports contain information to determine whether the server was restarted after the error occurred. Indeed, since NetSolve can be used over a very spread-out network, some *old* failure reports may very likely arrive after the server that failed has been restarted. The current policy ensures that a running server will not be seen as stopped by other servers. In other words, *a NetSolve can be stopped and restarted safely.*

When a server takes a failure into account, it marks the failed server in its data structures and does not remove it. A server will be removed only after a given time, and only if it has not been restarted.

5.2 Failure Robustness

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

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

5.3 Taking Failures into Account

When a failure occurs, the communication servers update their view of the NetSolve system. The communication servers keep track of the status of the remote hosts: reachable, or unreachable. They also keep track of the status of the NetSolve servers on these hosts: running, stopped, or failed. When a host is unreachable or a NetSolve server is stopped for more than 24 hours, the communication servers erase the corresponding entry in their view of the NetSolve system.
The communication servers also keep track of the number of failures encountered when using a computational server. Once this number reaches a limit value, the corresponding entry is removed. Therefore, if a computational server is poorly implemented, for instance because it calls a library incorrectly, it will eventually disappear from the system.

6 Linear Algebra Issues

NetSolve will deal with a large number of different computational problems. Here we focus on linear algebra problems.
We begin by giving a general definition for any arbitrary linear algebra problem. We define a problem as a 7-uple:

$$< name, m, v, s, M, V, S >,$$

where

- $name$ is a character string;
- m, v and s are the numbers of matrices, vectors, and scalars in **input** to the problem; and
- M, V and S are the numbers of matrices, vectors, and scalars in **output** to the problem,

To distinguish the different data types, we use the first character of the problem name, in the same way as it is done in LAPACK ([2]):

- S for real single precision
- D for real double precision
- C for complex simple precision
- Z for complex double precision

For instance, we could define a double-precision linear system solve as $<$ "$DAx = b$", $1, 1, 0, 0, 1, 0 >$. This definition allows to keep NetSolve as general purpose as possible in a linear algebra context. Some examples are given in 3.1.1.

6.1 Scientific Packages

NetSolve is able to use any scientific linear algebra package available on the platforms it is installed on, provided that the above formalism remains valid. This allows the *NetSolve administrator* not only to choose the best platform on which to install NetSolve, but also to select the best packages available on the chosen platform.
The current implementation of NetSolve at the University of Tennessee uses the BLAS (see [3], [4] and [5]) and LAPACK ([2]). These packages are available on a large number of platforms and are freely distributed. The use of ScaLAPACK ([6]) on massively parallel processors would be a way to use the power of high-performance parallel machines within NetSolve.

7 Performance

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

- Load balancing. Given all the computational resources available, NetSolve provides the user with a "best effort" to find the most suitable resource for a given problem.

- Simultaneous resources. Using the programming interfaces to NetSolve, the user can write a NetSolve application that has some parallelism. In Figure 4, we see the results of experiments conducted on a local network of SPARC workstations. The NetSolve program kept sending requests so that ten 600×600 eigenvalues problems were solved simultaneously over the network. We also added computational servers to the NetSolve configuration while running this program. Figure 4 shows the execution time for each problem for each experiment.

As expected, the problems are solved simultaneously on different servers, and the average execution time for one problem decreases when the number of computational servers increases.

8 Adding New Problems to a NetSolve Server

Part of our design objective was to ensure that NetSolve have an extensive application range. Thus, it had to be possible to add new problems to a computational server. Since it is unthinkable to have the NetSolve administrator modifying the NetSolve code itself for each new addition, we developed a simple tool to handle the task. The input for this tool is a configuration file describing each problem; the output is the actual C code of the computational process in charge of the problem solving. Thus, new problems can be added without having to be concerned about the NetSolve internal data structure.
In its first version, this pseudo-compiler still requires some effort from the NetSolve administrator. In fact, since any kind of library is supposed to be used within NetSolve, we cannot completely free the administrator from code writing. But we can provide him with a simple and efficient way of accessing the parameters to the problem. In particular, the function calls to the library have to be written in C, using a predefined set of macros. An example of the formal description of a problem is given in Appendix E.
More details on this set of macros and on the way to use them will be available as soon as the compiler reaches a stable development status.

9 Future Work

The initial NetSolve system is bound to undergo numerous modifications.

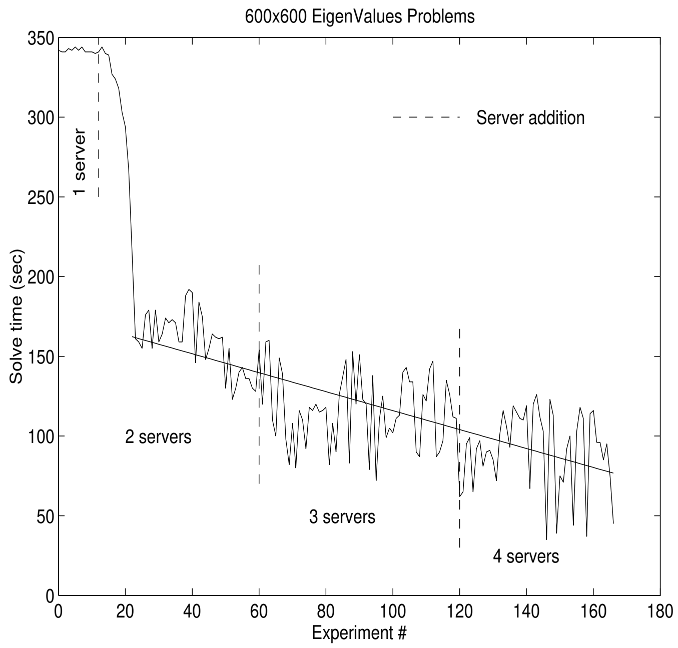

Figure 4: Simultaneous request to an evolving NetSolve system

The load-balancing strategy needs to be improved in order to change the "best guess" into a best "choice" as much as possible. The challenge is to come close to a best choice without flooding the network. Indeed, the danger is to waste more time computing this best choice than the computation would have taken in the case of a best guess only.

We also have to increase the number of interactive interfaces. For instance, we could write Maple and Mathematica interfaces, similar to the MATLAB one. Currently, we are thinking of providing the user with a HotJava interface on Solaris systems. Such an interface could be even easier to use than the MATLAB interface.

Another way to improve NetSolve is to extend it to a wider class of problems (i.e., beyond linear algebra). Indeed, any problem that can be expressed in terms of matrices, vectors, and scalars can be solved by a NetSolve computational server. For instance, it would be straightforward to interface NetSolve with an image-processing library.

All these improvements are intended to combine ease of use and performance, the main purpose of the NetSolve project.

A The GUI to NetSolve

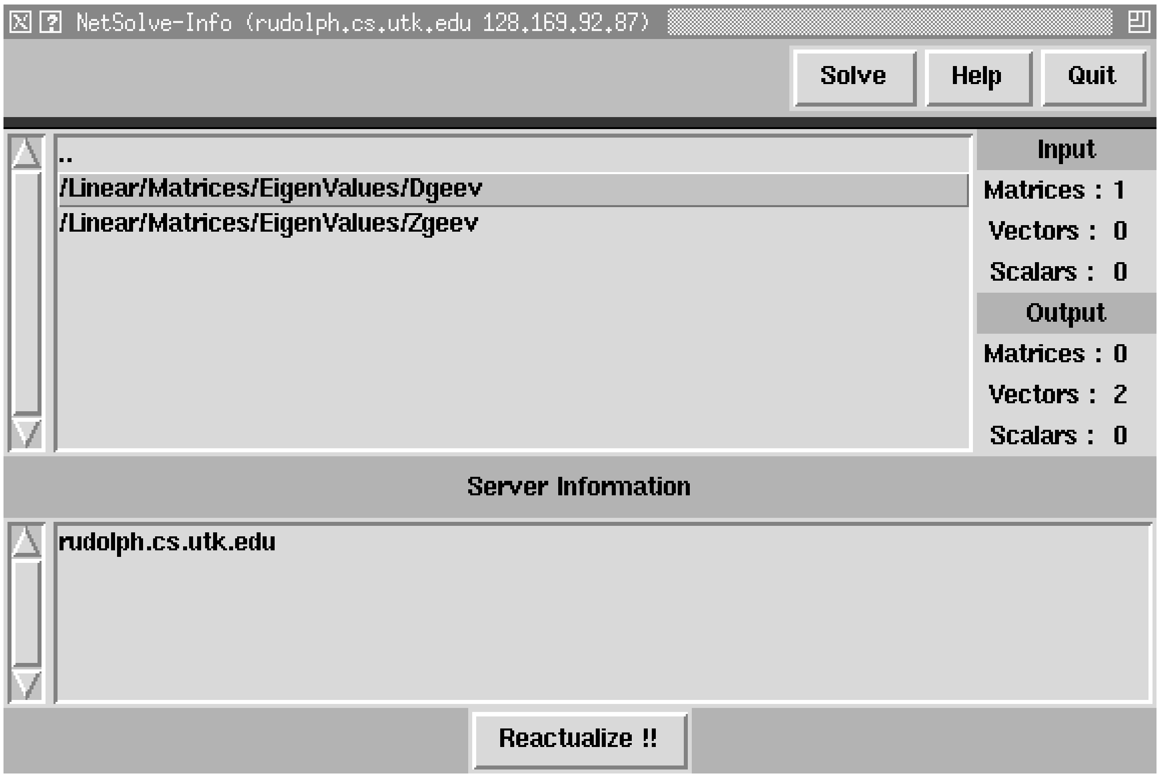

Figure 5: The NetSolve TK/TCL interface main window

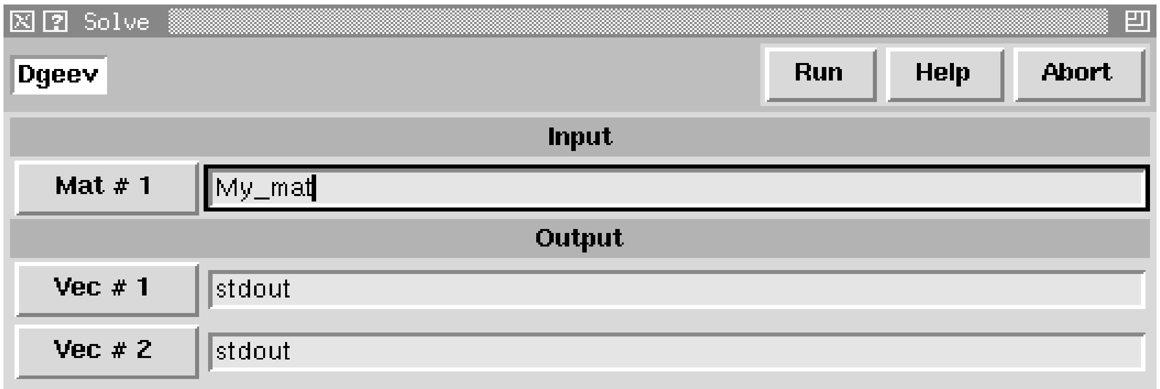

Figure 6: The NetSolve TK/TCL interface solve window

B Example: The NetSolve C Interface

```
...
...

double A[100*100];                  /* Matrix A                                     */
double Real[100],Imaginary[100];    /* real and imaginary parts of A's eigenvalues */
int request;                        /* NetSolve request number                      */
int is_finished;                    /* Flag giving the computation status           */
int n_b;

/**************************************/
/* Blocking call                      */
/**************************************/

request = netsolve("DEig",                /* Eigenvalues problem            */
              A,100,100,                  /* One matrix in input : A 100x100 */
              Real,&n_b,Imaginary,&n_b); /* Two vectors in output :         */
                                          /*   Real 100  and Imaginary 100   */

/**************************************/
/* Asynchronous call                  */
/**************************************/

request = netsolve_nb("DEig",A,100,100,
                     Real,&n_b,Imaginary,&n_b);
...
...    /* Some computations */
...

is_finished = netsolve_get(request,PROBE);     /* poll the previous request       */

...
...    /* Some computations */
...

is_finished = netsolve_get(request,WAIT);      /* poll in a blocking fashion      */
```

C Example: The NetSolve Fortran Interface

```
...
...
      INTEGER LDA,N
      PARAMETER(LDA = 100, N = 100)
      DOUBLE PRECISION A(LDA,N)
      DOUBLE PRECISION R(N),I(N)
      INTEGER REQUEST,BN
      INTEGER ISREADY

      CALL FNSINIT()

      ***********************************
      * Blocking Call                   *
      ***********************************

      CALL FNSOLVE('DEig',REQUEST,
     $             A,LDA,M,N,R,BN,I,BN)


      ***********************************
      * Asynchronous  Call              *
      ***********************************

      CALL FNSOLVE_NB('DEig',REQUEST,
     $                A,LDA,M,N,R,BN,I,BN)

...
...    * Some computations *
...

      CALL FNSGET(REQUEST,PROBE,ISREADY)
...
...    * Some computations *
...

      CALL FNSGET(REQUEST,WAIT,ISREADY)
```

D Adding resources to NetSolve

Let us first start a NetSolve agent on our network :

```
earth % netsolve_agent &
NetSolve Agent starting .....
NetSolve Agent running .....
earth %
```

Now that the agent is running, we can start resources. To start a computational server we need a configuration file containing information about the problems solvable by the new resource. We need also to specify the name of a machine running a NetSolve agent. This agent is the entry point into the system, and any agent can be chosen. Starting a resource on `saturn` is done below :

```
saturn % netsolve_resource -f file -a earth &
NetSolve resource starting .....
Problems initializing .....
Contacting agent on earth .....
NetSolve resource integrated in NetSolve system .....
saturn %
```

We can now check the status of the system with the interactive tool described in section 2.2

```
NetSolve earth > list sv

2 NetSolve servers in the system known by jupiter :
--------------
--> earth (128.145.32.234) (agent)
--> saturn (124.122.23.210) (resource)
NetSolve earth >
```

As shown above, it is very easy to add resources. The only pre-requisite is to know the name (or IP-address) of a machine running a NetSolve agent.

E *Formal* Description of the LAPACK Dgeev Function

```
#
# Dgesv
#
@PROBLEM                ** Beginning of a problem section                    **
@NAME
Dgesv
@DESCRIPTION
Linear system solve (from LAPACK)
@INPUT                  ** Number of matrices, vectors and scalars in input  **
1
1
0
@OUTPUT                 ** Number of matrices, vectors and scalars in output **
0
1
0
@COMPLEXITY             ** Complexity of the algorithm : 1xn^3               **
3
1
@LANGUAGE               ** Language of the target library                     **
FORTRAN
@FUNCTION
dgesv                   ** Name of the function to call                      **

@CODE                   ** Beginning of the CODE section                     **

 int nrhs = 1;
 int info;
 int *ipiv=NULL;
 if (*@nIV0@ != *@mIM0@)
   return BAD_DIMENSION;                     * Dimension mismatch        *
 *@nOV0@ = *@nIV0@;
 ipiv = (int *)malloc(sizeof(int)*(*@nIV0@));
 dgesv(@mIM0@,&nrhs,@IM0@,@mIM0@,                 * Call to the function *
         ipiv,@IV0@,@nIV0@,&info);
 @OV0@ = @IV0@;
 if (ipiv != NULL)
   free(ipiv);
 if (info >0)
   return NO_SOLUTION;                       * No solution to the problem *
 if (info <0)
   return LINEAR_FAILED;                     * Failure !!                *

@END_CODE               ** End of the CODE section                           **
```

References

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

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM Philadelphia, Pennsylvania, 2 edition, 1995.

[3] 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.

[4] 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.

[5] 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.

[6] J. Dongarra and D. Walker. Software libraries for linear algebra computations on high performance computers. *SIAM Review*, 37(2):151–180, 1995.