NetSolve's Network Enabled Server: Examples and Applications

Henri Casanova [*] Jack Dongarra[*] [†]

November 13, 1997

Abstract

The NetSolve project, underway at the University of Tennessee and Oak Ridge National Laboratory, allows users to access computational resources, such as hardware and software, distributed across the network. NetSolve provides a variety of interfaces so the user can easily perform scientific computing tasks without having any computing resource installed on his/her computer. There are many research issues involved in the NetSolve system, including 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 are emerging. In this article, we provide an overview of 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.

[*]Department of Computer Science, University of Tennessee, TN 37996, USA

[†]Mathematical Science Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

1 The NetSolve Project

1.1 Basics

As a result of advances in hardware, networking infrastructure and algorithms, computationally intensive problems in many areas can now be successfully attacked using networked, scientific computing. In the networked computing paradigm, vital pieces of software and information used by a computing process are spread across the network, and are identified and linked together only at run time. This is in contrast to the current software usage model where one acquires a copy (or copies) of task-specific software package for use on local hosts. 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 widely by Java applets within Web browsers. In the third paradigm, *remote computing*, the program resides on the server. The user's data is sent to the server, where the programs or numerical libraries operate on it; the result then is sent back to the user's machine. NetSolve uses this third paradigm.

Figure 1 depicts the typical layout of the system. 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 shown in the figure, the computational servers can be running on single workstations, networks of workstations that can collaborate for solving a problem, or MPP (Massively Parallel Processor) systems. The user gains access by using one of the NetSolve client interfaces. Through these interfaces, he can send requests to the NetSolve system asking for his 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. Once a server has been chosen, it is assigned the computation, uses its available numerical software, and eventually returns the results to the user. One of the major advantages of this approach is that the agent performs load-balancing among the different resources.

As shown on Figure 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 information about their different servers and allow access from any client to any server if desired. Suppose, for example, the set of computational resources span several local area networks and that users on each of these networks want to use NetSolve to perform scientific computations. It is then 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 either via the Internet or on an intranet, such as inside a research department or

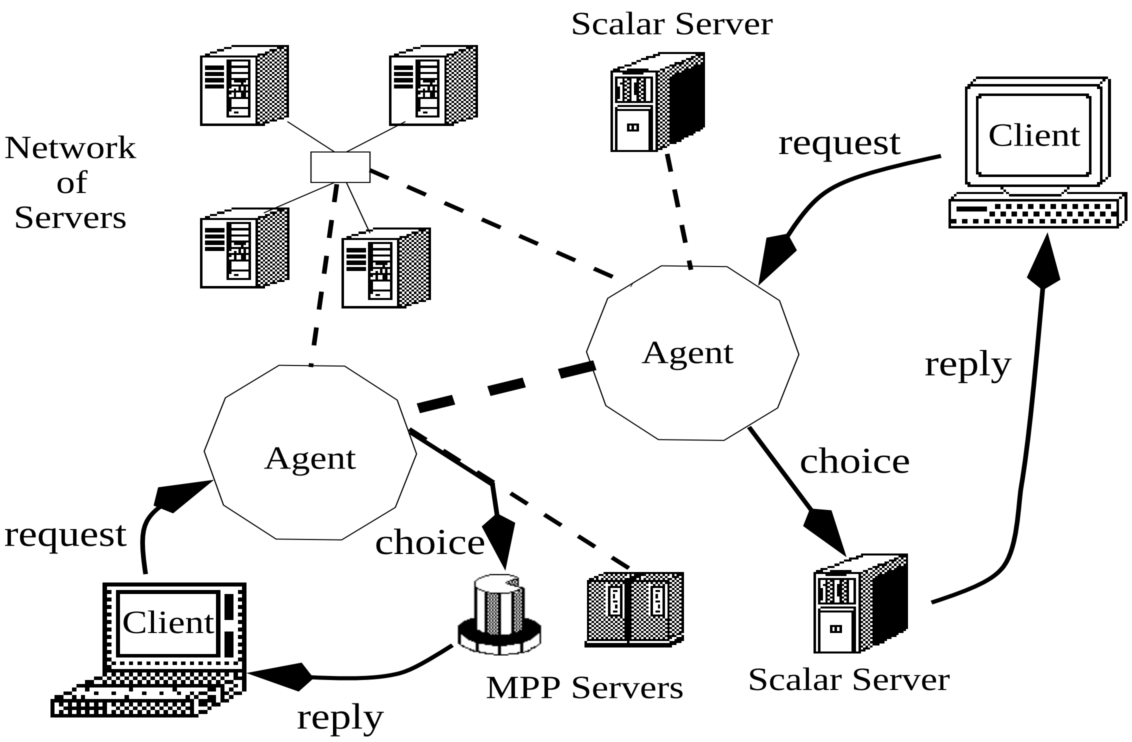

Figure 1: NetSolve's organization

a university, without participating in any Internet based 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.

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 1.4.

1.2 The Computational Resources

1.2.1 Challenges

When building the NetSolve system, one of the challenges was to design a suitable model for the computational servers. For the user to be able to invoke numerical software directly through our servers, three major features 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/functions. This is a critical point since we want to hide, as much as possible, the specifics of the underlying numerical software. In this way,

users will not need to 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 functionality to a computational server in an easy way. This would give the system the ability to 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, in a more general view, the system could dynamically handle installation and compilation itself, without any intervention from the user.

1.2.2 The Current Design

To make the implementation of such a computational server model possible, we have designed a general, machine-independent 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* which is used to describe each separate numerical functionality of a computational server. The description files written in this language can be compiled by NetSolve into actual computational modules executable on any UNIX or NT platform.

There are several advantages to this approach. 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 once. These files can then be exchanged by any institution wanting to set up servers. They can also be 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 increased.

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 (GUI) to handle the generation of the description files. The interface performs various error checking 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 complexity of the problem 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 very easily, as a result of 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 to us 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 those users setting up NetSolve computational servers can create their description files directly from within Web browsers. These files can 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 such a 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 nearly immediate. The idea would then be to add a complementary repository containing NetSolve description files to a regular software repository like Netlib [1].

1.2.3 Existing Resources

A number of description files have been generated for the following numerical libraries: ARPACK [2], FitPack [3], ItPack [4], MinPack [5], FFTPACK [6], LAPACK [7], BLAS [8, 9, 10], QMR [11], Minpack [5] and ScaLAPACK [12].

NetSolve computational servers providing access to these libraries are currently running at the University of Tennessee and at other locations world-wide. Real-time information on the running servers can be found on the NetSolve web-page located at:

`http://www.cs.utk.edu/netsolve`.

These numerical libraries cover several fields of computational science; linear algebra, optimization, fast fourier transforms, etc. Some of the subroutines in these libraries require the user to supply a function, for example to evaluate a function to be minimized. The current version of the NetSolve software handles user-supplied functions in a way described in [13].

1.3 The Client Interfaces

A major concern in designing NetSolve was to provide several interfaces for a wide range of users. NetSolve can be invoked through C, Fortran, Java, as well as on Matlab. In addition, there is a Web-based Java GUI which allows problems to be and solved remotely. Another concern was keeping the interfaces as simple as possible. For example, there are only two calls in the MATLAB interface, and they are sufficient to allow users to 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 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" [13]. In Section 1.4.4, we show a utilization example of NetSolve that takes advantage of our agent-based strategy.

1.4 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.

1.4.1 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 task of the NetSolve agent. Since the computational servers use the same framework to contribute software to the system (see Section 1.2.2), it is possible for the agent to maintain a database of different numerical functionalities available to the users.

The protocol is fairly straightforward. Each time a new server is started, it sends an application 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 and detects 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 is a candidate for being used by a client. The next section explains how the agent might make such a decision.

1.4.2 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. For each user request, the agent determines the set of servers that can handle the computation and makes a choice between all the possible resources. 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. dimensions of the matrices for a linear algebra computation), etc. Resource-specific information 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 represents the load of the server's host, the network delays, and transmission rates to contact that host. The network performance is actively estimated by the agent by continuously averaging samples of the network delays between the hosts. The strategy for the load of the servers is different; the computational servers actively notify the agent of their workload fluctuations when they deem it necessary. Rationale and further detail on these protocols can be found in [14].

1.4.3 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. This is the reason why NetSolve does not try to impose any control on the different resources. Though this approach is flexible, it requires NetSolve to 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 has a list 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 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 one of these servers has been successfully contacted, the numerical computation starts. If the contacted server fails during the computation, then another server is contacted and the computation restarts. This whole process is transparent to the user. Each time a computational server malfunction (server unreachable, server stopped, failure during computation, etc.) 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, then the user is notified that the computation can not be performed at that time.

1.4.4 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 one we are describing here provides information about the kind of typical gain a user can obtain by invoking NetSolve, as well as a measure of the NetSolve overhead. In this experiment, the user is using MATLAB on a Sun workstation (Sparc 5) to perform several matrix multiplications. The size of the matrices is 800 by 800, and the user wants to perform up to 16 consecutive multiplications. A NetSolve system is available and consists of 7 computational servers. These servers run on Sun workstations (Ultra 1's). It should be noted that the results of these experiments would be the same if instead of one single user, several users were sending requests to the NetSolve system.

Figure 2 shows the total execution times for various number of matrix multiplications in two cases: (i) the user is using MATLAB, (ii) the user is using the asynchronous calls of the MATLAB interface to NetSolve. 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. The "Intranet" curve is for a client located on the same local area network than the servers. The "Close Internet" curve corresponds to a client at the Oak Ridge National Laboratory. The "Continental Internet" is for a client at the University of California, Berkeley. Finally, the "Overseas Internet" is for a client located at the Danish Technical University in Denmark. The matrix multiplication takes roughly 34 seconds on the client machine, using MATLAB. It takes approximatively 20 seconds on the server machines, using the BLAS.

In the case where MATLAB is used directly, the total execution time increases linearly with the number of operations performed at a rate equal to the execution time of one matrix multiplication. This is of course expected since the multiplications are executed one after the other. One can see that using NetSolve leads to much better execution times, except when the client is located overseas. In fact, transferring the data and the result overseas is much more costly than performing a matrix multiplication. In all the other cases, it is always better to use NetSolve, even if only one multiplication is needed.

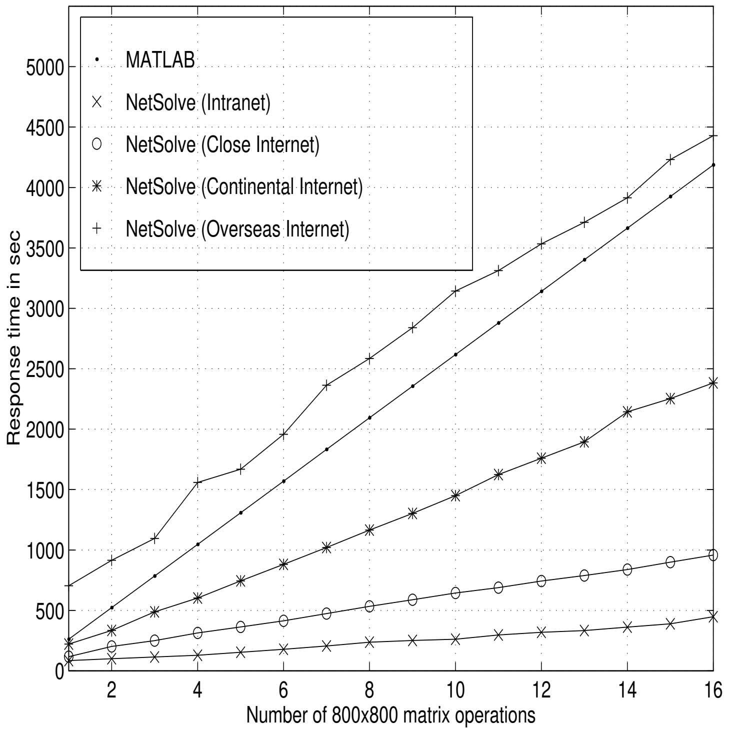

Figure 2: Multiple 800x800 matrix multiplications

The agent is locating powerful machines, if any, on behalf of the user and schedules the user's computations on those machines to minimize execution time. In the setting of these experiments, all the servers were identical and their workloads were identical as well. The agent was therefore compelled to schedule the computation in what appears to be a round-robin fashion.

For these advantages to be worthwhile, the user should be required to change as little of his program as possible. The MATLAB code to perform a matrix multiply is given below:

```
c = a * b
```

and an exactly equivalent NetSolve code using a blocking call could be:

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

However, to achieve the speed-ups of 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 by using NetSolve. This is even more true about the C and Fortran interfaces. Even more strikingly, the same performance can be achieved from a Java program calling the NetSolve Java API, since the numerical computation will be performed with compiled Fortran or C on some other platform.

1.5 Current and Future Directions

Agent-based computing is a promising strategy. NetSolve will evolve into a more elaborate system in the future and a major part of this evolution is to take place within the agent. The changes will address different parts of the agent concept. We highlight some of the modifications in what follows.

As the system increases both in number of users and resources, it will be more and more 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 [15], RCDS[16]) and implementations are starting to become available. Such services would provide a good basis for a metacomputing project like NetSolve and would relieve the NetSolve developers from taking naming responsibilities.

NetSolve will eventually need to provide a user-accounting feature so that realistic bounds can be imposed on resource usage. We could, for example, 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. In the current version of the software, every computational server can be started in an access restriction mode. The restrictions are described by maximum numbers of simultaneous pending requests from different domain names (down to different IP-addresses), not taking the actual users into account. This is a reasonable strategy for now, but a more flexible and practical 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 example, students would not have the ability to run large computations, whereas researchers could have full access. Administrative authorities could also customize their own accounting policy and put bounds on the usage of their resources. These bounds could be in terms of CPU time, or MBytes on hardware resources, or in terms of number of requests. Two universities or national laboratories could then allow each other to use every resource with, however, a "preference" for the local users to use the local resources. The NetSolve agent could then be the primary actor in a sophisticated accounting mechanism.

Another step to be taken in the NetSolve project concerns data encryption and compression. Indeed, they become almost mandatory for any realistic metacomputing project. It is at this time still uncertain how NetSolve will provide these services. The preliminary investigations imply the use of already existing metacomputing toolkits like the Globus/Nexus [17] project, for example, to achieve such goals.

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. There are many difficulties in providing a uniform performance metric that encompasses any type of algorithmic and hardware considerations in a metacomputing setting, especially when different numerical resources, or even entire frameworks are integrated into NetSolve. Such integrations are described in the following sections.

2 An Interface to the Condor System

2.1 Overview of Condor

Condor [18, 19, 20], developed at the University of Wisconsin, Madison, is a high throughput computing environment that can manage very large collections of distributively owned workstations. 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. The environment is based on a layered architecture that enables it to provide a powerful and flexible suite of Resource Management services to sequential and parallel applications. Condor has been ported to most UNIX platforms. Condor views the owners of the resources as the key holders to the success of a High Throughput Computing environment. It therefore pays special attention to the rights and sensitivities of the workstation owners. It is the owner of each and every workstation in the collection who defines the conditions under which the workstation can be allocated by Condor to an external. Condor jobs that consist of a single process are automatically checkpointed and migrated between workstations as needed to ensure eventual completion.

A brief description of Condor's software architecture follows. A Condor pool consists of any number of machines, of possibly 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, mouse/keyboard activity, etc.) and decides if it is available to run a Condor job. The *schedd* keeps track of all the Condor jobs that have been submitted to the machine. One of the 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 *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 been started, it is periodically checkpointed. It can then be interrupted and migrated within the Condor pool until completion. This organization is partly depicted in Figure 3. More details on the Condor system and the software layers can be found in [18, 19, 20]. In the next section, we explain how a Condor pool is being used as the back-end to a NetSolve computational resource.

2.2 A Condor Pool as a NetSolve Resource

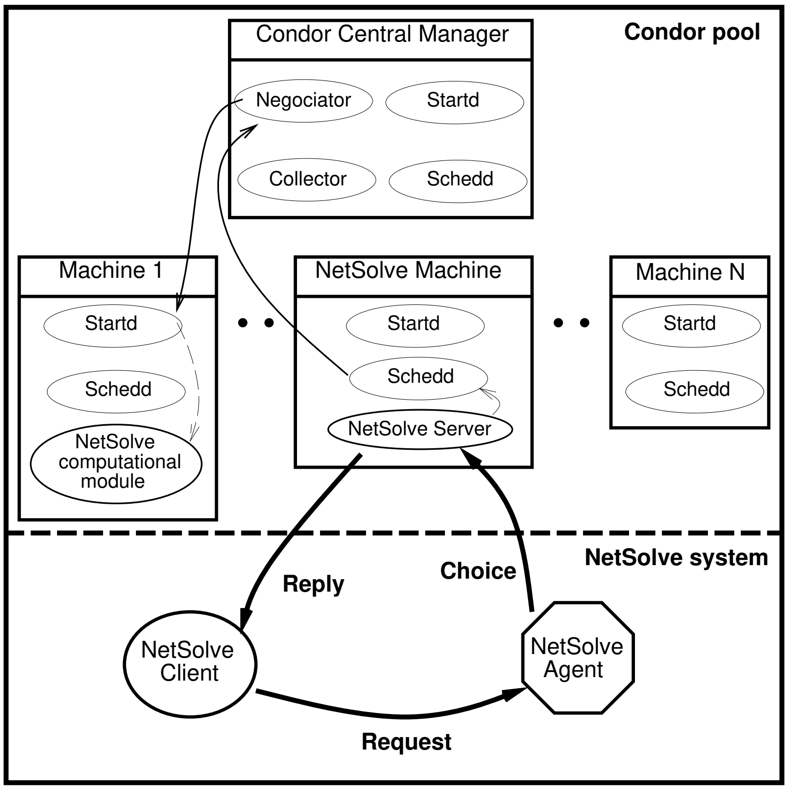

Figure 3: NetSolve and Condor

2.2.1 Motivation

Interfacing NetSolve and Condor is a very natural idea. NetSolve provides remote easy access to computational resources through multiple, attractive user interfaces. Condor allows users to harness the power of a pool of machines while using otherwise wasted CPU cycles. The users at the consoles of those machines are not penalized by the scheduling of Condor jobs. If the pool of machines is reasonably large, it is usually the case that Condor jobs can be scheduled almost immediately. This could prove to be very interesting for a project like NetSolve. Indeed, NetSolve servers may be started so that they grant local resource access to outside users. Such servers are permanently available a the University of Tennessee for instance. Interfacing NetSolve and Condor could then give priority to the local users and provide underutilized only CPU cycles to NetSolve users. Such an interfacing is described in the next section.

2.2.2 Implementation

Figure 3 shows how an entire Condor pool can be seen as a single NetSolve computational resource. The Condor pool consists of several machines. The Central Manager is represented at the top of the figure. As stated before, it 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 of those machines 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 be migrated among the machines in the pool. When the results of the numerical computation are obtained, the NetSolve server transmits that result back to the client.

The actual implementation of the NetSolve/Condor interface was made easy by the Condor tools provided to the Condor user. In the NetSolve server code, it consisted of replacing a call to `execv()` by a Condor call. However, the restrictions that apply to a Condor jobs concerning system calls were difficult and required quite a few changes to obtain a Condor-enabled NetSolve server. A major issue however 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 at this point unclear how these metrics can be directly applied. A possible approach could be to have the NetSolve server running in the Condor pool to collect statistics on the pool behavior. These statistics could then be used by the NetSolve agent to compute predictions of job execution times in that pool. Finding the appropriate prediction technique will be at the focus of the next step in the NetSolve/Condor collaboration.

3 Integrating Parallel Numerical Libraries

3.1 Motivation

Integrating parallel packages into NetSolve will allow a user on a PC or workstation to access MPP systems to perform large computation. This access can be extremely simple and the user may not even be aware that he is 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.

NetSolve views a computational resource as a NetSolve server running on some platform. The specifics of that platform are totally hidden from the user. Section 2 already described how a platform can in fact be a set of machines contributing to the computations requested by a user. In the case of Condor, the computation is sequential and the computational process may migrate among the different machines to take advantage of unused CPU cycles. There is another obvious way in which several machines can be considered as a single computational resource to NetSolve; make those machines participate in a parallel computation. Enabling NetSolve to use parallel numerical libraries would be a very interesting and natural development. A user could still use the simple NetSolve interfaces to access the power of MPPs or networks of workstations to perform large computations. In the next section, we describe the first steps in integrating the ScaLAPACK library into a NetSolve server.

3.2 Integrating Parallel Software Packages into NetSolve

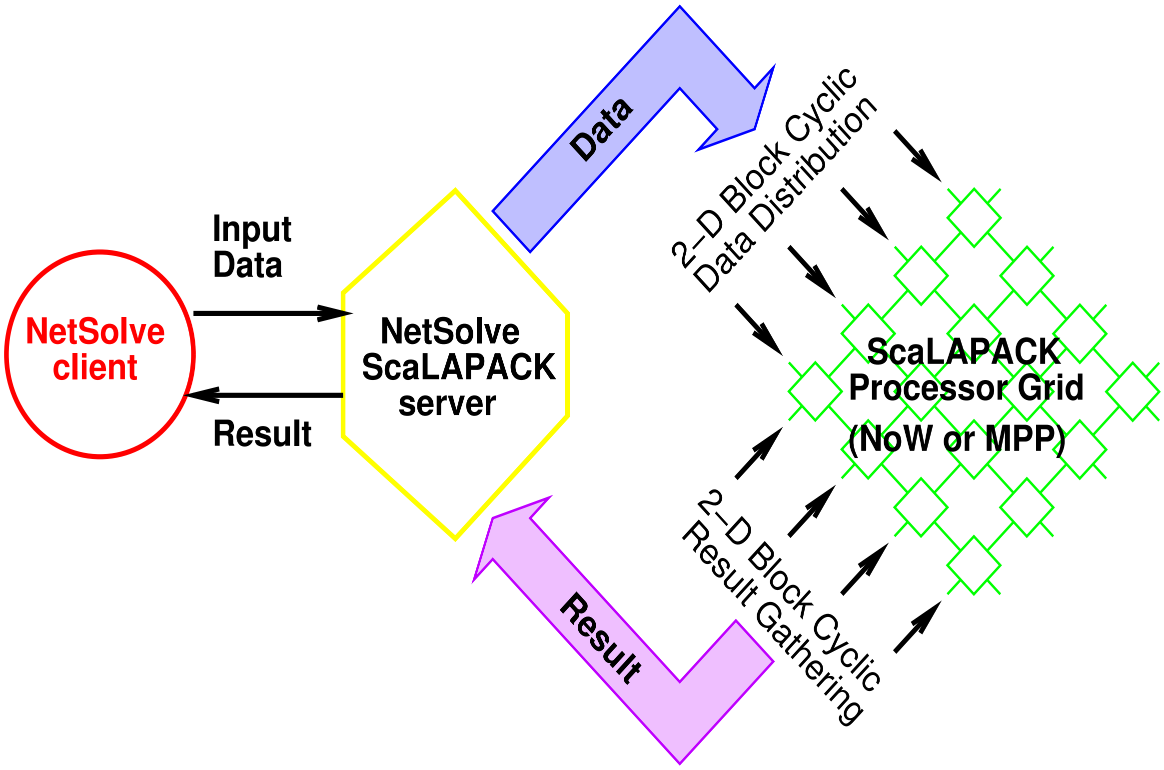

Figure 4: The ScaLAPACK NetSolve Server Paradigm

ScaLAPACK (Scalable Linear Algebra Package) is a library of high-performance linear algebra routines for distributed-memory message-passing MIMD computers as well as networks of workstations supporting PVM [21] or MPI [22]. ScaLAPACK was developed at the University of Tennessee, Knoxville, Oak Ridge National Laboratory and the University of California, Berkeley. It is a continuation of the LAPACK [7] project, 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. The goals of ScaLAPACK include efficiency, scalability, reliability and portability. ScaLAPACK views the underlying multi-processor system as a rectangular process *grid*. Global data is mapped to the local memories of the processes in that grid assuming specific data-distributions. For performance reasons, ScaLAPACK uses the two-dimensional block cyclic distribution scheme for dense matrix computations. Inter-process communication within ScaLAPACK is done via the BLACS (Basic Linear Algebra Communication subprograms) [23, 24]. The BLACS is implemented in terms of the available native message-passing facilities and is specially designed for linear algebra applications. All the details on ScaLAPACK can be found in the latest edition of the User's Guide [12].

Figure 4 is a very simple description of how the NetSolve server has been customized to use the ScaLAPACK library. The customized server receives data input from the client in the traditional way. Depending on the implementation environment, the server either already has access to a set of processors, or must request those processors from the system. Once the required number of processors is available, the NetSolve server uses BLACS calls to set up the ScaLAPACK processor grid. ScaLAPACK requires that the data already be distributed among the processors prior to any library call. This is the reason why each user input is first 2-D block cyclic distributed in that grid when necessary. The server can then initiate the call to ScaLAPACK and wait until completion of the computation. When the ScaLAPACK call returns, the result of the computation is usually available on the processors and is 2-D block cyclic distributed as well. The server then gathers that result and sends it back to the client in the expected format. 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 client can use MATLAB on a PC and issue a simple call like `[x] = netsolve('eig',a)` and have an MPP system use a high-performance library to perform a large eigenvalue computation. We have designed a prototype of the customized server running on top of PVM [21] or MPI [22]. There are many research issues arising with integrating parallel libraries in NetSolve. First, the agent currently needs to perform performance predictions for parallel algorithms running in various distributed systems. Such predictions will be much more involved than the ones performed by the agent at the moment. Furthermore, the agent may 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 2-D block cyclic distribution. This choice usually depends on the nature and the 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 will have to 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 accounting.

4 NetSolve and Ninf

4.1 A Brief Overview of Ninf

Ninf[25], developed at the Electrotechnical Laboratory, Tsukuba, Japan, is a global network-wide computing infrastructure project which allows users to access computational resources including hardware, software, and scientific data distributed across a wide area network with an easy-to-use interface. Ninf is intended not only to exploit high performance in network parallel computing, but also to provide high quality numerical computation services and accesses to scientific database published by other researchers. Computational resources are shared as Ninf remote libraries and are executable at a remote Ninf server. Users can build an application by calling the libraries with the Ninf Remote Procedure Call, which is designed to provide a programming interface similar to conventional function calls in existing languages, and is tailored for scientific computation. In order 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. Ninf also interfaces with existing network service such as the WWW for easy accessibility. Clearly, NetSolve and Ninf bear strong similarities both in motivation and general design. Allowing the two systems to coexist and collaborate should lead to promising developments, and the next section describes some preliminary work that has been done in this view.

4.2 A Gateway Between Ninf and NetSolve

The content of this section results from documents exchanged between the NetSolve and Ninf development teams during the early stages of this collaboration [26, 27]. A collaboration between the two projects seemed natural. However, some design issues prevent an immediate seamless integration of these two systems. First, the Ninf MetaServer and the NetSolve Agent, even though similar in intent, have different philosophies. The NetSolve Agent, as seen in Section 1.4, is really a resource broker, whereas the Ninf MetaServer is a *proxy*. Second, the user interfaces of the two systems are very different. The Ninf interface structure is based on C-like calling sequences. NetSolve, on the other hand, uses an abstract and language-independent I/O description. These are the two major sources of difficulties, but there are numerous details that contribute to make a seamless integration a non-trivial task (data transfer protocols, data types/structures supported, etc.).

In order to overcome these issues, the Ninf team started developing two *adapters*: a NetSolve-Ninf adapter and a Ninf NetSolve-adapter. Thanks to those adapters, Ninf clients can use computational resources administrated by a NetSolve system and vice-versa. This first implementation of the adapters is written in Java, and is explained in Figure 5 and 6.

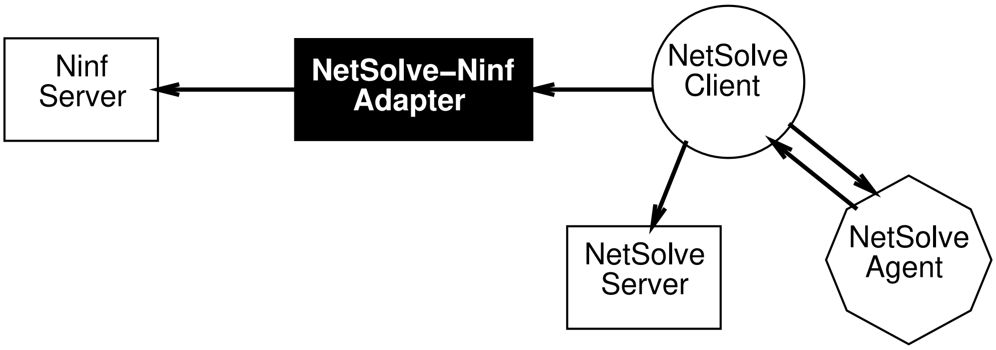

Figure 5: Going from NetSolve to Ninf

Figure 5 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. When a NetSolve client sends a request to the agent, it can then be told to use the NetSolve adapter. The adapter performs protocol translation, interface translation, and data transfer in such a way that the NetSolve client does not realize that it is not interacting with a NetSolve server. The adapter then asks a Ninf server to perform the required computation and returns the result to the user.

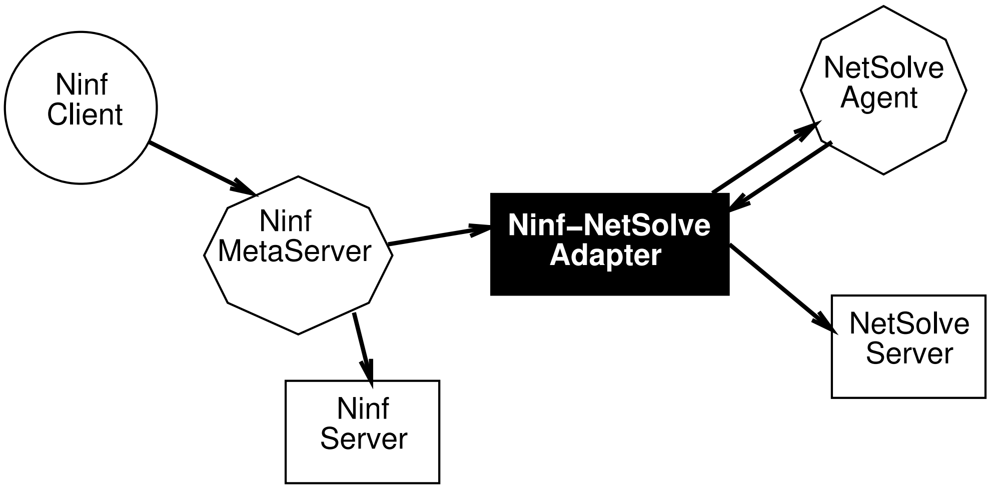

Figure 6: Going from Ninf to NetSolve

Figure 6 is very similar to Figure 5. The NetSolve-Ninf adapter can be 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 previously mentioned, 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 client.

There are several advantages of using such adapters. NetSolve and Ninf are still young projects and as such are evolving rapidly. It would therefore be unrealistic to keep modifying the design of either one of the projects for the sake of collaboration. By contrast, just updating the adapters to reflects the evolutions of NetSolve or Ninf seems perfectly reasonable. Some early implementation evaluations tend to show that using either system via an adapter causes acceptable overheads, mainly due to additional data transfers. Those first experiments appear encouraging and will definitely be extended to effectively enable an integration of NetSolve and Ninf.

5 Extending ImageVision by the Use of NetSolve

In this section, we describe how NetSolve can be used as a building block for a general purpose framework for basic image processing.

5.1 Integrating the ImageVision Library into NetSolve

This project is under development at the Institute for Computer Graphics (ICG) at Graz University of Technology, Austria. The scope of the project is to make basic image processing functions available for remote execution over a network. The goals of the project include 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. Let us first describe briefly the ImageVision Library (IL) [28]. IL is an object-oriented library written in C++ by Silicon Graphics, Inc. (SGI) and shipped with their workstations. It contains typical image processing routines to access, manipulate, display, and store image data. It is also multi-threaded and can therefore make use of multiprocessor machines and other special graphic hardwares. ImageVision has been judged quite complete and mature by the research team at ICG and seems therefore a good choice as 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) and is described in [29].

5.2 NetSolve as an Operating Environment for ImageVision

The reasons why NetSolve has been a first choice for such a project are diverse. First, NetSolve is easy to understand, use, and extend. For example, it very simple to call NetSolve at the client level. Second, NetSolve is freely available. Third, NetSolve provides language binding to Fortran, C, and Java. And finally, NetSolve's agent-based design allows load monitoring and balancing among the available servers. New NetSolve computational modules corresponding to the desired image processing functionalities will be created and integrated into the NetSolve servers in the way explained in Section 1.2.2. The servers will then grant access to these functionalities via all the NetSolve user interfaces. A big part of the project at ICG is to build a Java GUI to IL. This will be done with the NetSolve Java API now available.

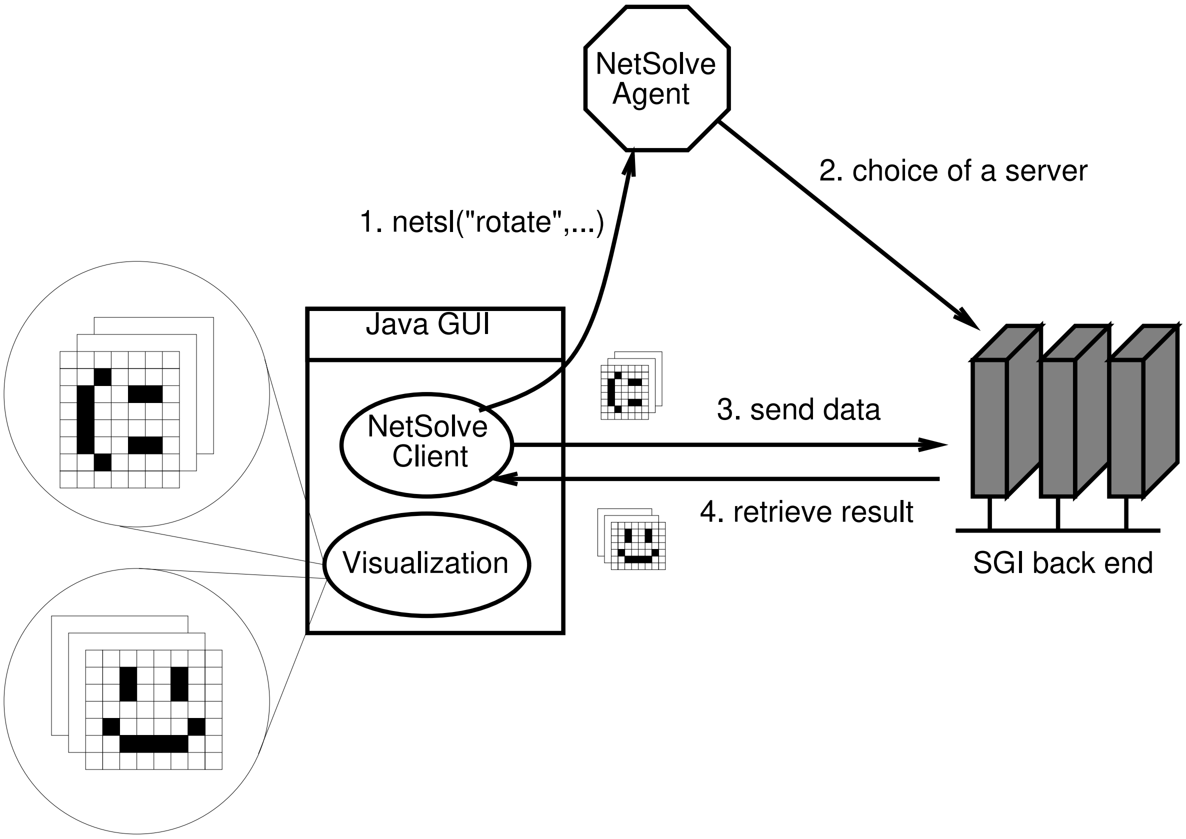

Figure 7: ImageVision and NetSolve

Figure 7 shows a simple example of how ImageVision can be accessed via NetSolve. A Java GUI can be built on top of the NetSolve Java API. As shown on the figure, this GUI offers visualization capabilities. For computations, it uses an embedded NetSolve client and contacts SGI servers that have access to IL. The user of the Java GUI does not realize that NetSolve is the back end of the system, or that they are using a SGI library without running the GUI on a SGI machine! The protocol depicted on Figure 7 is of course simplistic. In order to obtain acceptable levels of performance, the network traffic needs to be minimized. There are several ways of approaching this problem. It would for instance be possible to keep a "state" in the server, meaning that some server always keeps the most recent image objects in memory. It would also be possible to pack several operations in one single request. Finally, a possibility is for the client to give only references to images (URLs for instance) instead of image data. All these possibilities are under investigations at the moment.

Integrating image processing functionalities in NetSolve will undoubtedly raise some new issues about NetSolve's design and abilities. We will be considering the integration of HTTP and FTP protocols within NetSolve, on-the-fly compression/decompression of user data, and security considerations.

6 Conclusion

The scientific community has long used the Internet for communication of email, 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.

We have discussed throughout this paper how NetSolve can be customized, extended, and used for a variety of purposes. We first described in Section 2 how an entire Condor pool can become a NetSolve computational resource. Somewhat similarly, Section 3 describes how a parallel machine (MPP or network of workstations) can also be used as a single resource by NetSolve via the use of the ScaLAPACK library. Those two experiments are extending the range of use of NetSolve in different ways. They also raise new research issues. One of these issue, and perhaps the most important, concerns the agent resource management strategy. Indeed, 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 next discussed the NetSolve/Ninf collaboration. Ninf is a project somewhat similar to NetSolve, at least in its intent, and we described how the two projects can be bridged to provide a set of resources usable by either NetSolve of Ninf clients, as well as the preliminary work that has already been done in this view. Making such metacomputing projects interact is quite a challenge due to differences in designs, protocols, and user interfaces but is one step further toward a metacomputing standard. In Section 5, we gave an example of an entire application that uses NetSolve as an building block. This application is an extension of the ImageVision library to a more general image processing framework that allows remote access from a network. Network enabled servers, and NetSolve in particular, have helped in developing such a framework and NetSolve has been selected as the operating environment. 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.

References

[1] 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/.

[2] R. Leboucq, D. Sorensen, and C. Yang. *ARPACK Users Guide*. 1997.

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

[4] 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.

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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*.

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

[16] K. Moore, S. Browne, J. Cox, and J. Gettler. Resource Cataloging and Distribution System. Technical Report CS-97-346, Computer Science Dept, University of Tennessee, Knoxville, TN, Jan. 1997.

[17] I. Foster and K Kesselman. Globus: A Metacomputing Infrastructure Toolkit. In *Proc. Workshop on Environments and Tools*. SIAM, to appear.

[18] M. Litzkow, M. Livny, and M.W. Mutka. Condor - A Hunter of Idel Workstations. In *Proc. of the 8th International Conference of Distributed Computing Systems*, pages 104–111. Department of Computer Science, University of Winsconsin, Madison, June 1988.

[19] M. Litzkow and M. Livny. Experience with the Condor Distributed Batch System. In *Proc. of IEEE Workshop on Experimental Distributed Systems*. Department of Computer Science, University of Winsconsin, Madison, 1990.

[20] J. Pruyne and M. Livny. A Worldwide Flock of Condors : Load Sharing among Workstation Clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.

[21] 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.

[22] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI : The Complete Reference*. The MIT Press Cambridge, Massachusetts, 1996.

[23] J. Dongarra and R. van de Geijn. Two dimensional basic linear algebra communication subprograms. Technical Report CS-91-138, Computer Science Dept, University of Tennessee, Knoxville, TN, 1991. Also LAPACK Working Note #37.

[24] R.C. Whaley and J. Dongarra. A user's guide to the BLACS v1.1. Technical Report CS-95-281, Computer Science Dept, University of Tennessee, Knoxville, TN, 1995. Also LAPACK Working Note #118.

[25] 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.

[26] H. Casanova. Enabling Collaboration between NetSolve and Ninf. May 1997.

[27] H. Najada, S. Matsuoka, and S. Sekigushi. Bridging Ninf and NetSolve. September 1997.

[28] G. Eckel, J. Neider, and E. Bassler. *ImageVision Library Programming Guide*. Silicon Graphics, Inc., Mountain View, CA, 1996.

[29] M. Oberhuber. Integrating imagevision into netsolve. Available at http://www.icg.tu-graz.ac.at/mober/pub, October 1997.