CONCURRENCY—PRACTICE AND EXPERIENCE
Concurrency: Pract. Exper. 2001; **00**:1–13 *Prepared using* cpeauth.cls *[Version: 2001/03/05 v2.01]*

Innovations of the NetSolve Grid Computing System

Dorian C. Arnold[1,*], Henri Casanova[2] and Jack Dongarra[1]

[1] *1122 Volunteer Blvd., Suite 413, Computer Science Department,
University of Tennessee, Knoxville, TN 37923-3450*
[2] *9500 Gilman Dr., Computer Science and Engineering Department,
University of California, San Diego, La Jolla, CA 92093-0114*

SUMMARY

The NetSolve Grid Computing System was first developed in the mid 1990s to provide users with seamless access to remote computational hardware and software resources. Since then, the system has benefitted from many enhancements like security services, data management faculties and distributed storage infrastructures. This article is meant to provide the reader with details regarding the present state of the project, describing the current architecture of the system, its latest innovations and other systems that make use of the NetSolve infrastructure.

KEY WORDS: Grid Computing, Distributed Computing, Heterogeneous Network Computing, Client-Server, Agent-based Computing

INTRODUCTION

Since the early to mid 1990s, the distributed computing domain has been a hot-bed of research as scientists explored techniques to expand the boundaries of scientific computing. In this era, the term Grid Computing [1] was coined to describe a fabric, analogous to the electrical power grid, that would uniformly and seamlessly channel computational services to clients who "plug in" to the Grid. The NetSolve system, from the University of Tennessee's Innovative Computing Laboratory, was one of the earlier Grid systems developed. NetSolve's first motivation was to address the ease-of-use, portability and availability of optimized software libraries for high-performance computing. Incidentally, this mode of use, providing client-users with remote access to software services, has become NetSolve's niche and separates it from other Grid systems that require users to have software "in hand." The NetSolve system enables users to solve complex scientific problems remotely, by managing networked computational resources

[*] Correspondence to: 1122 Volunteer Blvd., Suite 413, Computer Science Department, University of Tennessee, Knoxville, TN 37923-3450. Email: darnold@cs.utk.edu

Received 31 January 2000
Revised 12 May 2000

and using intelligent scheduling heuristics to allocate resources that will efficiently satisfy requests for service. Virtually any software package can be transformed into a service available to NetSolve client-users.

This article discusses major developments made since NetSolve's initial deployment. We describe NetSolve's architecture and implementation, the Grid services NetSolve provides, specialized features of the system and examples of NetSolve usage. We conclude with a discussion of future research directions. The scope of this article does not include details regarding how to use NetSolve – such information can be found in [2].

THE NETSOLVE SYSTEM

The Architecture

An instance of a *NetSolve Grid* is set of (possibly heterogeneous) computer hosts accessible via some networking infrastructure, like an ethernet or even a modem. The system uses a client/agent/server model and is available for all popular variants of the UNIX operating system, and parts of the system are available for the Microsoft Windows platforms. The major components of the NetSolve system are the NetSolve agent, an information service and resource scheduler, the NetSolve server, a networked resource that serves up computational hardware and software resources, and the NetSolve client libraries, that allow users to instrument their application code with calls for remote computational services. We term a program enhanced with calls to the NetSolve system as NetSolve-enabled. Figure 1 shows the infrastructure of the NetSolve system and its relation to the applications that use it. NetSolve and systems like it are often referred to as Grid middleware. The shaded parts of the figure represent the NetSolve components, and it can be seen that NetSolve acts as a glue layer that brings the application or user together with the hardware and/or software services needed.

Implementation

The NetSolve system is implemented using the C programming language, with the exception of the thin upper layers of the client application programming interfaces (API) that serve as an environment specific interface to the NetSolve system. (This layer primarily manipulates input objects and converts them into a NetSolve specific format.) The system's components use TCP/IPv4 sockets and a NetSolve-specific application layer protocol to communicate with each other. Heterogeneity is supported by implementing a "handshake" transaction that initiates each communication session during which computer hosts determine whether they understand the same data format (endianness, data sizes, etc.) When an incompatibility is detected, Sun's XDR [3] protocol is used to encode local data to a globally-understood format, allowing each host to decode from this external data representation to its local format. The system is primarily meant to be "open" in the sense that root privileges are not required to install system components, and, once installed, one can easily donate services to the community at large. However, for those that require a more controlled system, NetSolve incorporates Kerberos V5 [4] protocols to allow for client-server authentication and access control.

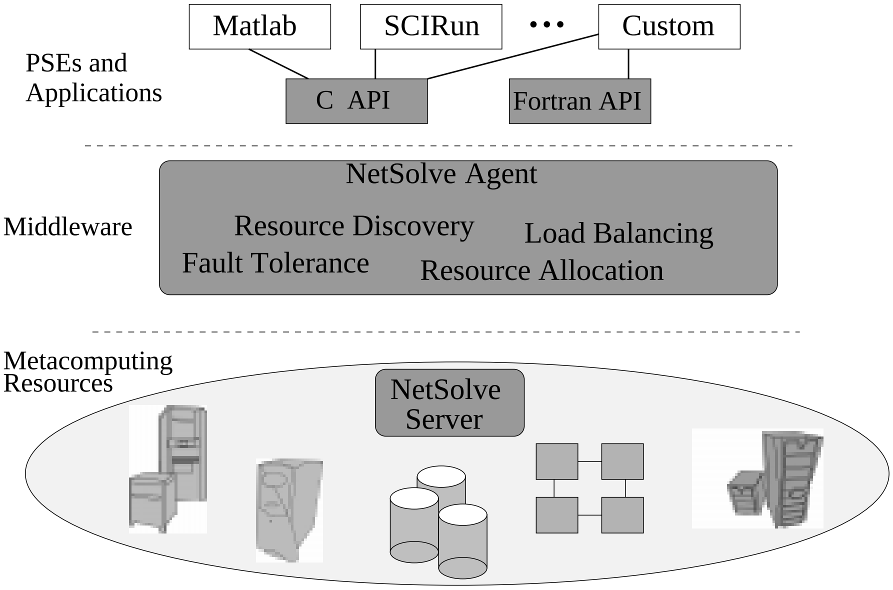

Figure 1. An architectural overview of the NetSolve system.

Managemental Issues

A NetSolve pool will most naturally consist of independent workstations or clusters of workstations. We do support the use of supercomputer class machines (MPPs and SMMPs) as servers, but we do not interface to batch queues like LoadLeveler [5] or PBS [6]. Instead, on these classes of machines parallel jobs are launched in an interactive manner. Each NetSolve component is a self-supported entity that can therefore be managed independently. A server remains running even after an agent to which it was registered goes down. As the agent is the entry-point into the NetSolve system, such a scenario renders a server useless. Nevertheless, it persists until another instance of the agent re-enters the system and assumes the duty of its predecessor.

A number of command line and web-based tools exist to allow NetSolve system administrators (maintainers of running NetSolve servers and agents) to detect and modify the system configuration. These tools allow for querying the configuration and modifying it by terminating agents or servers, or re-configuring servers with new or different services, etc.

Prepared using cpeauth.cls

THE NETSOLVE CLIENT

NetSolve provides a functional programming model (based on RPC) in which the client API is used to pass NetSolve objects to and from services as inputs and outputs – though not an object-oriented system, NetSolve maintains all its components in object data-structures. Accordingly, the NetSolve object types are `MATRIX`, a two-dimensional array, `SPARSEMATRIX`, a two-dimensional array stored in a compressed row storage format, `VECTOR`, a one dimensional array, `SCALAR`, `STRING`, an array of characters, `STRINGLIST`, an array of strings, `FILE`, and `UPF`, a user-provided function. The data elements of these objects may be single or double precision integers, real numbers or complex numbers, or characters. The NetSolve client interface supports both synchronous and asynchronous calls. The synchronous or blocking call transfers control to the NetSolve system which invokes a request on behalf of the client and blocks until the results are available. At this point, it returns control to the user application layer with the appropriate output results. The asynchronous or non-blocking version of the call makes the request to the NetSolve system and returns immediately. The client program is given a handle to the request that is used to probe for the status of the request (RUNNING, COMPLETE, FAILED) and gather the output if/when they are available.

Client Interfaces

NetSolve supports client program development in a variety of programming environments. The two major classifications of these environments are the interactive environments that dynamically interpret programming commands or instructions and compiled programming interfaces that statically compile instructions into a binary executable. We describe interfaces to our system from both environments.

Interactive environments

Interactive environments are usually easier, especially for the non-programmer, often relieving the user from details like variable declaration, memory allocation, and other mundane tasks. NetSolve currently supports APIs for both the Matlab [7] and Mathematica [8] environments. NetSolve enhances these environments by expanding the numerical functions available to the environment and allowing for increased performance by executing code remotely on more efficient machines. Furthermore, if the remote server is a workstation cluster or parallel machine, the application is able to leverage multiple processors as compared to the single local processor available to the Matlab or Mathematica environment. The asynchronous interfaces also allow for a simpler level of parallelism by allowing for simultaneous requests to the NetSolve system. The discussion below focuses on the Matlab interface, but for each Matlab feature or function we demonstrate, there is an analogous functionality in Mathematica.

The left side of figure 2 shows an example of using MATLAB to solve a linear system of equations using the blocking call. This script first creates a random 100 × 100 matrix, A, and a vector b of length 100. The call to the `netsolve()` function returns with the solution. The (hidden) semantics of a NetSolve request are:

```
>> A = rand(100); b = rand(100,1);
>> x = netsolve(’Ax=b’,A,b);
```

```
>> A = rand(100); b = rand(100,1);
>> request = netsolve_nb(’send’,’Ax=b’,A,b);
>> x = netsolve_nb(’probe’,request);
.... Not Ready Yet
>> x = netsolve_nb(’wait’,request);
```

Figure 2. Examples of calling NetSolve from within the Matlab environment in blocking (left) and non-blocking (right) fashions.

1. Client contacts the agent for a list of capable servers.
2. Client contacts “best” server and sends input parameters.
3. Server runs specified service on input parameters.
4. Server returns output parameters or error status to client.

The right hand-side of the same figure 2 shows the same computation performed in a non-blocking fashion. In this case, the first call to `netsolve_nb()` sends a request to the NetSolve agent and returns immediately with a request identifier. One can then either *probe* for the request or *wait* for it. Probing always returns immediately, either signaling that the result is not available yet or, if available, stores the result in the user data space. *Waiting* blocks until the result is available and then stores it in the user data space.

Other functions are provided, for example, to query the status and configuration of a NetSolve pool, the set of problems available, and the function signatures (definitions of the number and types of input/output parameters) of the software services available.

Programming interfaces

The programming interfaces for which NetSolve APIs have been developed are FORTRAN and C. Unlike the interactive interfaces, programming interfaces require some programming effort from the user. The user is responsible for defining and allocating arrays for input and output variables. In figure 3, the above Matlab example is demonstrated using the C programming language and the NetSolve blocking call.

Specialized Client Features

Now that we have covered the basics of the NetSolve client libraries, we move to discuss key modifications developed to enhance the usability, efficiency and flexibility of the interfaces to the NetSolve system.

Client Proxies

The architecture of NetSolve has been modified to include client proxies. A proxy is a separate process that resides on the client host whose purpose is to negotiate with the underlying meta-

```
double A[100*100]; /* Matrix A */
double b[100]; /* Right Hand Side b */
double x[100]; /* Solution Vector */
int status; /* NetSolve request status */

status = netsl("Ax=b",
A,100,100, /* Coefficient and dimensions */
b, /* RHS */
x); /* Solution Vector, x */
```

Figure 3. Example 'C' code program shows details of NetSolve's blocking interface.

computing infrastructure for services on the client program's behalf. The primary reasons for the addition of a proxy to the NetSolve framework are:

- **Lightening the client libraries**
  By separating the interactions with meta-computing resources from the NetSolve client interface, the client library becomes much more lightweight. One advantage of this scenario is that it increases the uniformity with which the supported client interfaces can be developed. In other words, this modularity means that whenever a feature is added or an enhancement made at the proxy level or below, it only need be implemented once and immediately becomes available to all interfaces.
- **More flexibility on the client side**
  Since the proxy is a separate process with its own thread of control, it is able to interact with meta-computing resources, independently of client interaction. This makes it possible to perform tasks like queries of various Grid information services, even before the client makes a request, and cache results locally so they are immediately available when needed.
- **Integration with other systems**
  One of the philosophies of the NetSolve project is to leverage existing services whenever feasible. Having a proxy negotiate for meta-computing resources on behalf of the client means that different proxies can negotiate for different types of services. So far, we have implemented proxies to negotiate for Globus [9] services and, of course, the standard NetSolve services. Other systems like Condor [10] or Legion [11] can be integrated in a similar fashion.
- **Supporting more client languages**
  With a standard interface between the client and proxy, it is possible, especially for third party developers, to easily add new language support to the NetSolve system. They would simply write libraries that interface the NetSolve proxies from their language of choice, allowing programs of that language to become NetSolve-enabled.

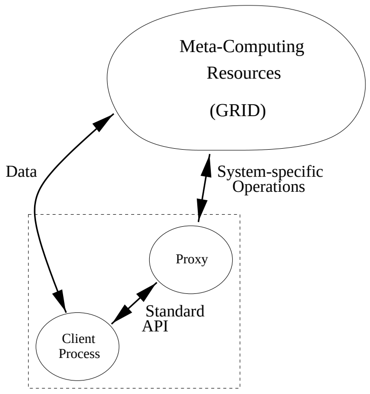

Figure 4. Proxy Architecture

Figure 4 shows the interaction between the client process, proxy and underlying Grid system. The client libraries interact with the proxy via a standard API, and the proxy interacts with the meta-computing system using system-specific mechanisms. The NetSolve proxy, for instance, uses the agent to discover services, contacts the appropriate server and establishes a session with that server. The server receives input data from the client, performs the requested service and returns output data to the client process. Each client process instantiates a new proxy and when security services are enabled, there is no delegation of credentials from client to proxy. Instead, the connection between the client and server, established for the transmission of data, is authenticated. This would be the point at which unauthorized access would be detected and refused.

Data Persistence

The left side of figure 5 illustrates the typical transactions that take place during a series of NetSolve requests made by the sequence of calls:

```
...
netsl(‘‘command1’’, A, B, C);
netsl(‘‘command2’’, A, C, D);
netsl(‘‘command3’’, D, E, F);
...
```

What is relevant in this example is that parameter `A` is shared as an input for the first and second requests. Also, output parameters `C` and `D` serve as inputs for subsequent requests. This is exactly the type of scenario that the work described in this section tries to exploit.

We have explored a technique we call *request sequencing* in an attempt to maximize request throughput by minimizing data transmission between NetSolve components. This interface allows the user to group two or more regular NetSolve requests into a sequence. The system then analyzes the input and output parameters of the sequence to determine when parameters

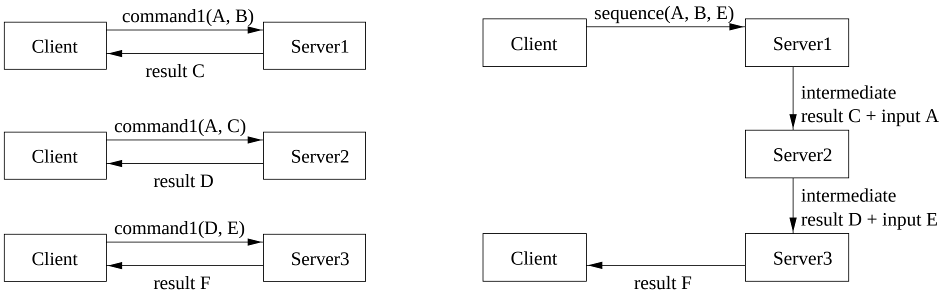

Figure 5. Client-server interactions during a typical request scenario (left) and during a "request sequence" (right).

recur and keeps those parameters persistent at or near the server(s) that are servicing the requests. More specifically, we construct a directed acyclic graph (whose nodes represent computational modules and arcs represent the data dependencies of those modules) and schedule this DAG for execution. Our hypothesis is that this reduction in data traffic will yield enough performance improvements to outweigh the overhead of the DAG construction and make sequencing worthwhile. The right side of figure 5 shows the reduced data flow between client and server components that occurs when the sequencing mode is employed.

[12] discusses this interface and gives some experimental results that were achieved using this data persistence technique. The results support our theories that this is a very good way to optimize the use of Grid resources, especially when bandwidth is of the essence, data sets are very large, or both.

Distributed Storage Infrastructures

A Distributed Storage Infrastructure or DSI [13] is a technology that allows a program to manage data stored remotely. The Internet Backplane Protocol [14] (IBP) is an example DSI that has been incorporated into NetSolve in an effort to logistically store data in storage depots convenient to NetSolve servers that will run computations on them. This integration allows users to allocate, destroy, read and write data objects to remote storage devices, via IBP, and then point NetSolve servers to these devices to find data to use in computations. The user can thus run computations on remote data and retrieve only pertinent portions of the output.

The NetSolve API has been extended with a set of functions to create, destroy, open, close, read and write DSI storage. We modeled the API after the 'C' STDIO library. However, we did tune the read and write calls to take advantage of NetSolve system-specific characteristics. The read and write calls are designed to deposit and extract these objects from DSI storage. Figure shows a sample programming code that makes calls to a version of the interface (simplified for this document.) The 'C' struct `NS_DSI_FILE` encapsulates information about the remote storage server and remote file object being allocated, while `NS_DSI_OBJECT` contains

```
int client_program(){
NS_DSI_FILE * rfile;
NS_DSI_OBJECT * robject;
int *data1, size_data1, status;

...
rfile = ns_dsi_open(“machine.domain.edu”, “write”);
robject = ns_dsi_write_matrix(rfile, data1, size_data1);

status = netsolve(“solve_matrix”, robject, rhs);
...
}
```

Figure 6. Sample ‘C’ code program shows details of NetSolve’s DSI interface.

information about specific objects present within these remote files. In essence, a call to `ns_dsi_open()` allocates a remote file returning a handle or `NS_DSI_FILE *`. A call to any of the `ns_dsi_write_*()` variants will transfer data to the remote storage specified by the given handle. These handles can then be used directly in NetSolve requests or by corresponding calls to `ns_dsi_read_*()` variants. When used in NetSolve requests, the NetSolve system resolves the handles to appropriate datasets and uses them either to locate input on which to perform computational analysis or to store the results of computation.

The primary technique used in this first iteration of development is an instance of supply-side caching where the application at the client layer of the system is the supplier of data to the server which acts as the data consumer. The system is manually configured to strategically place storage servers near pools of computational servers as in the experiments of the next section. We also export to the user-layer a low-level interface that the user must use to dictate where data should be stored and where computations should be performed.

The major modifications made to the NetSolve system involve the expansion of NetSolve’s object model to include representations for remote data objects and files. The API functions described above, apart from the obvious, also put and later extract information into and from these structures. NetSolve maintains a File Allocation Table (FAT) that records the status of allocated remote files and objects much like that used by operating systems to keep track of STDIO files. The reference values of the `NS_DSI_OBJECT`s and `NS_DSI_FILE`s are used as the keys by which these objects are cataloged in the FAT. When NetSolve requests are made, input and output references in the calling sequence are checked against the keys of the FAT to see if they represent a remote object. (If not found, they are assumed to be referring to local data, in-core or on disk). The NetSolve system protocols accommodate remote data by sending data handles to servers which the servers use to obtain data from their stored locations. This implementation allows the NetSolve system to leverage DSI storage without modifying the standard NetSolve functions for computational requests.

Task Farming

This interface addresses applications that have simple task-parallel structures but require large number of computational resources. Monte-Carlo simulations and parameter-space searches are some popular examples of these applications which are called `task farming` applications. Within the NetSolve system, we designed and built an easily accessible computational framework for task farming applications. As the middleware, NetSolve becomes responsible for the details of managing the Grid services – resource selection and allocation, data movement, I/O, and fault-tolerance.

The task farming interface allows the user to make a single call to NetSolve, requesting multiple instances of the same problem. Rather than single parameters, the user passes arrays of parameters as input/output and designates how NetSolve should iterate across the arrays for the task farm. The main challenge in this effort is scheduling. Indeed, for long running farming applications it is to be expected that the availability and workload of resources within the server pool will change dynamically. [15] discusses the design and validation of the NetSolve task farming interface. It also presents an adaptive scheduling algorithm used by the task farming interface to assign tasks to the server resources.

Transparent Algorithm Selection

Through NetSolve, users are given access to complex algorithms that solve a variety of types of problems, one instance being linear systems solvers. All solvers, however, are not built alike; depending on the characteristics of the matrix being solved some perform poorly and others not at all. NetSolve has incorporated a large number of solver algorithms from a variety of packages like PETSc [16] and Aztec [17]. We have further created an interface that allows the user to generically call a "LinearSolve" routine which transparently analyzes the input matrix and determines which algorithm to use based on input characteristics. The system recognizes matrix features like size, shape, sparsity, symmetry, and bandwidth dimensions and decides which of the available solver packages is most appropriate to correctly and efficiently solve the system.[18] further discusses this interface and the heuristics and decisions that are involved in the algorithm selection process. This interface allows the non-expert user to properly and efficiently use solver algorithms without climbing the steep learning curve that would otherwise be involved to learn the intricacies of a specific software package (or its auxiliaries, such as MPI or BLAS). This feature exemplifies the ease-of-computing that is only one of the benefits of using a system like NetSolve.

SERVER AND SERVICE SPECIFICATIONS

Server Specification

A computer host becomes a NetSolve server when it is configured to run the NetSolve server daemon. After an initialization process, the server daemon listens to a TCP/IP socket (bound on a port published to the NetSolve agent during initialization) for incoming requests for service. We now discuss the different modes of configuration and other components of the server.

The configuration of a server

When a server is initiated, it first obtains a *raw* performance rating using the LINPACK benchmark. This rating is measured in terms of Kflop/s and is reported to the agent for use in the scheduling heuristics discussed in the *Section Resource Allocation in NetSolve.* During the initialization process, the server also reads a `server_config` file. This file allows a NetSolve administrator to specify several details: the agent host that the server will register to, its workload threshold (explained below), scratch space for temporary files, number of simultaneous service requests permitted, and the number of processors located on the server node. Most importantly, this file allows one to specify which software services a server will be able to provide. We soon discuss what comprises a software service and how such a service is executed by the server.

The workload manager

An important part of the server is the workload manager. This component monitors the workload of the server host via the UNIX `uptime` utility and reports this value to the agent.

Figure 7 shows the scheme used to manage the workload broadcasts. Let us consider a computational server, M, and an instance of the agent, C. M broadcasts its workload periodically. In Figure 7, we call *time slice* the delay between workload evaluations of 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. However, 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 rebroadcasted. 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 measured, the workload manager decides that the next value to be broadcast should be different enough from the last value reported – in other words, outside this confidence interval. In Figure 7, the workload is broadcast three times during the first five time slices. (As an added heuristic, after a certain specified interval, the workload manager is forced to broadcast its workload, regardless of the value.)

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 or confidence interval that is too narrow causes the workload to be assessed too

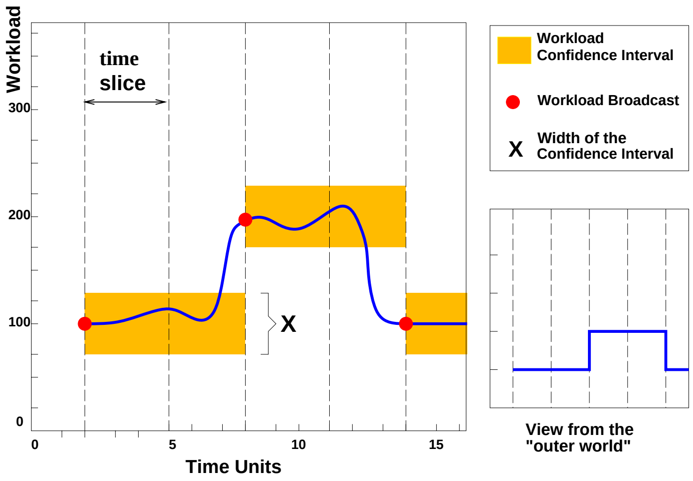

Figure 7. Workload Policy in NetSolve

often or a lot of useless workload broadcasting. If these same parameters are too wide, the workload manager will be slow to recognize major changes in the workload or these significant changes will not be broadcasted. Choosing an effective time slice and confidence interval helps to make the workload information maintained by the agent as accurate as possible, so that the resource scheduling heuristics (covered in the discussion of the NetSolve agent) are reasonable. Regardless of the intervals chosen, whenever a workload is reported to the agent, a measurement of the network bandwidth and latency is also evaluated and reported to the agent. The connection measured is that between the agent and server. A NetSolve pool is best configured with an agent on every local network from which a potential client will reside so that this measure is also a fair evaluation of the network between the client and server hosts once a client request is instantiated.

Service Specification

To keep NetSolve as general as possible, we need to find a formal way of describing a problem. Such a description must be carefully chosen, since it will affect the ability to interface NetSolve with arbitrary software.

From a client's perspective, a problem is a 3-tuple: $< name, inputs, outputs >$, where

- *name* is a character string containing the name of the problem
- *inputs* is a list of input objects
- *outputs* is a list of output objects

The possible types and elements of an object have been defined in Section *NETSOLVE CLIENT*. These items are specified in a problem description file (PDF) that also specifies libraries containing the implementations of any underlying functions or services being interfaced by NetSolve and a code that uses some specialized, pre-defined macros to extract data elements from the NetSolve input objects, pass them to the underlying service functions, and place output results in NetSolve output objects. The PDF also allows a user to specify the computational complexity of the algorithm, with respect to the size of the data inputs. A tool called the `code generator` parses this PDF to create actual C code that is then linked with NetSolve auxiliary libraries (that transfer data to/from client programs and handle other generic service tasks) and those libraries specified by the PDF to create a service executable.

Integrated scientific services

Using the PDF facility, NetSolve has integrated a variety of scientific packages. These PDFs are distributed with the current implementation of NetSolve: BLAS [19] [20] [21], LAPACK [22], ScaLAPACK [23], ItPack [24], PETSc [16], AZTEC [17], MA28 [25], SuperLU [26], and ARPACK [27]. When efficient, we include the actual numerical library with NetSolve, in the other cases, these packages are available for a large number of platforms and are freely distributed.

The Server/Service Binding

Upon server initialization, the server discovers which problems (PDFs) it will be configured to service via the aforementioned `server_config` file. It reads in a description for every problem (name, number and types of inputs and outputs, etc.) and maintains this information in its core memory. It also locates the appropriate service executables and registers its `problem list` and each problem's description with the agent specified in the `server_config` file. Whenever there is an incoming request from a client, after the server determines its willingness to service that request, it "execs" the appropriate service program. This program generally opens a network connection to receive input data from the client, calls the underlying service function to run the computation, and then returns output data to the client.

Access Control Mechanism

Interaction in a Grid or any distributed environment ultimately demands that there are reassuring mechanisms in place to restrict and control access to computational resources and sensitive information. In the latest version of NetSolve, we have introduced the ability to generate access control lists which are used to grant and deny access to the NetSolve servers. We use Kerberos V5 [4] services to add this functionality to NetSolve, as it is one of the most trusted and popular infrastructures for authentication services. Using Kerberos, the "administrator" of the server identifies authorized clients using their Kerberos principal, or id. He places his list of clients in a file that is readable only by the user-id that will be used to run the NetSolve server. The only other interaction needed to "kerberize" the server is to create a principal that is used to identify the NetSolve service within the Kerberos realm. Every other involvement needed is just as it would be in non-Kerberos mode. This feature has been implemented such that kerberized and non-kerberized client and server components can gracefully interact with each other. Kerberized servers simply deny service to non-authenticated clients, while clients configured to send authentication credentials will only do so upon demand by a kerberized server.

Network Weather Service CPU Sensors

NetSolve servers must be able to attain and report information regarding their workload so that the agent can determine which servers represent the best choice to service a request. The `workload manager` performs this task, however, the Network Weather Service [28] is a Grid service utility with a component, the CPU sensor, that does this as well. The section below on *AGENT INTERACTIONS* discusses our integration of NWS forecasters and motivates our use of the NWS CPU sensors as well. There are several features that make this NWS component attractive to the NetSolve project. The NetSolve workload manager was a simple way for the system to implement a feature that at the time was not available any other way; NWS is a project whose purpose is to make this type of service available, and thus much time and expertise has been invested to make sure the reports are as accurate as possible. Secondly, NWS sensors interact with a separate process, called a `memory` to store the data being collected. This process need not be run on the host being monitored, therefore, we can conveniently place the NWS memory on the host running the NetSolve agent (and NWS forecaster) making the information readily available when a forecast is needed. Finally, it offers the uniformity of using the NWS components together. Otherwise, we would have to modify the NetSolve system so it is somehow able to feed the relevant information to the NWS forecaster that helps the agent allocate resources.

AGENT INTERACTIONS

Though the NetSolve system is merely based upon the concept of remote procedure call (RPC), its uncomplicated user interaction allows one to easily overlook the other features it entails and consider it *just* an RPC system. NetSolve is a robust environment that is able to discover, maintain and control a heterogeneous environment of vastly distributed computational resources. The purpose of the agent is to be that point in the system from which to manage and access the resources. As an information service, the agent maintains a comprehensive view of the status of all NetSolve server components and the interactions they may be having with clients. It keeps track of the software capabilities of the servers and is able to direct client requests to appropriate servers.

Resource Allocation in NetSolve

The performance model

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

$$p = \frac{P \times n}{\frac{w}{100} + 1}$$

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 estimate this time on every machine M in the NetSolve system. We split the time T into T_n, the time to transmit data to/from M, and T_c, the time to perform the computation on M. The time T_n can be computed by knowing the following:

1. network latency and bandwidth between the local host and M, and
2. size of the data to be transmitted.

The computation of T_c involves knowledge of the

1. size of the problem,
2. complexity of the algorithm to be used, and,
3. performance of M as defined by our performance model

The client-dependent parameters, specified service and sizes of data to be transmitted, are included in the problem request sent to the agent. The **static** server-dependent parameters, service algorithm complexity and raw performance, are reported to the agent during the server initialization process. Finally, **dynamic** server-dependent parameters, load of the server host and network measures, are accounted for by reports to the agent from the workload manager discussed above.

Load Balancing

The NetSolve system does not explicitly attempt to balance the load on its computational servers. As mentioned above, it incorporates scheduling heuristics that attempt to optimize service turn-around time. A server has the capability to service simultaneously as many requests as the operating system will allow it to fork processes, but the administrator can specify the maximum number of requests it is willing to service at one time; requests for services received once this limit is met will be rejected. Similarly, a server can specify its workload threshold and, once again, requests received after this threshold is attained will be refused. Carefully using these threshold heuristics, a NetSolve administrator can maintain a balanced system.

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 catastrophic failure. 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.

Failure detection

NetSolve detects and classifies two major classifications of failures: *HOST_ERROR* and *SERVER_ERROR*. A *HOST_ERROR* is a “hard” failure that denotes permanent inability to contact a host or a server on a host. A *SERVER_ERROR* is a “soft” failure that denotes a running server has experienced some failure that will not necessarily prevent it from being of future use.

*HOST_ERROR*s are detected when a component fails to establish a network connection with a server. Failures may occur at different levels of the NetSolve protocols. Generally they are due to a network malfunction, to a server disappearance, or to a server failure. The connection might have failed or have reached a time-out before completion. *SERVER_ERROR*s are experienced either when there is a failure in the communication protocol, or when the server explicitly denotes that it experienced a failure while trying to service a request.

As a heuristic, the *workload manager* is forced to broadcast a report within a certain time period. The agent detects a possibly terminated server when it fails to broadcast its workload within this period.

Taking failures into account

Ultimately, all failures are reported to the agent who determines what actions to take. Since *HOST_ERROR*s are permanent failures, the agent removes suspect servers and their problems from the database. When the agent takes a *SERVER_ERROR* into account, it marks the failed server in its data structures and does not remove it. *SERVER_ERROR*s are almost always due to malformed input or output parameters that cause errors in the transmission of data or “service failures” like segmentation violations, etc. Therefore, a policy decision was made to presume that service implementations have been properly tested and that a *SERVER_ERROR* is always based on an improperly programmed client.

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 the NetSolve agent receives a request for a problem to be solved, 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 to the agent for the same problem if it encounters a *HOST_ERROR*. If at the end of the list no server has been able to answer, the client returns with an error status denoting “no available server.” Since a *SERVER_ERROR* is seen by the system as caused by the format of the client call, on such an error, no retries are invoked, and an error of “server/service error” is reported.

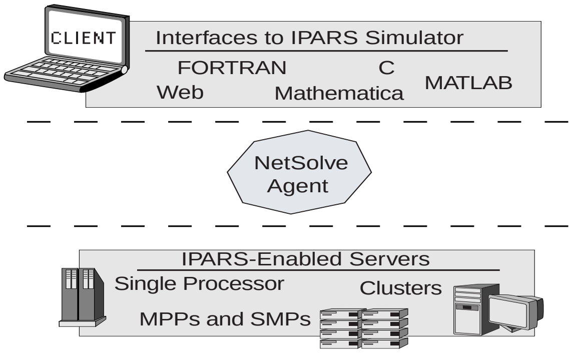

Figure 8. Integrating a fluid flow simulator with NetSolve.

Network Weather Service Forecasters

When allocating resources, the agent's main goals is to choose the best-suited computational server for each incoming request We have employed the use of the Network Weather Service (NWS) [28] to help us gather the information necessary to make these decisions. The NetSolve agent calls upon the services of NWS forecasters to predict future availability of server hosts based upon previously collected availability data (see Section *NWS CPU Sensors*). The forecasters are actually an interface to a variety of prediction modules which take as input a time series of measurements and returns a prediction of the next value. High levels of accuracy are achieved since a prediction error of each module is calculated, and the prediction with the lowest error is used. Previously, NetSolve would look at the last recorded value of a servers status and use that to represent the resources that a server would have available to service a request. NWS allows NetSolve to do a much better job of "guessing" what a server's availability will be, especially when servers experience high variations in workload.

NETSOLVE USAGE

Sub-surface Modeling

The implicit parallel accurate reservoir simulator, IPARS, developed at the University of Texas' Institute for Computational and Applied Mathematics, is a framework for developing parallel models of sub-surface flow and fluid transport through porous media. It simulates single phase (water only), two phase (water and oil) or three phase (water, oil and gas) flow through a multi-block 3D porous medium. IPARS can be applied to model water table decline due to overproduction near urban areas, or enhanced oil and gas recovery in industrial applications.

A NetSolve interface to the IPARS system (Fig. 8) allows users to access IPARS (including post-processing of the output to create animated images that exhibit the variations of concentration, pressure, etc. of relevant fluids) [29]. The interface is primarily used from handy machines like laptop computers to run real-time simulations on clusters of workstations that allow for much quicker execution. IPARS runs primarily on LINUX; NetSolve makes it readily accessible from any platform. In addition, we have created a problem solving environment (PSE) interfaced by a web browser which one can use to enter input parameters and submit a request for execution of the IPARS simulator to a NetSolve system. The output images are then brought back and displayed by the web browser. This interaction shows how the NetSolve system can be used to create a robust grid computing environment in which powerful modeling software, like IPARS, becomes both easier to use and administrate.

Cellular Microphysiology

MCell is a general Monte Carlo simulator of cellular microphysiology which uses Monte Carlo diffusion and chemical reaction algorithms in 3D to simulate the complex biochemical interactions of molecules inside and outside of living cells. MCell is a collaborative effort between the Terry Sejnowski lab at the Salk Institute, and the Miriam Salpeter lab at Cornell University.

NetSolve's farming interface is very well suited to MCell's needs. One of the central pieces of that framework is a scheduler that takes advantage of MCell input data requirements to minimize execution turn-around time. This scheduler is part of an AppLeS project[30] at the University of California, San Diego. The use of NetSolve isolates the scheduler from the resource-management details and allows researchers to focus on the scheduler design.

Nuclear Engineering

The goal of this project is to develop a prototype environment for the Collaborative Environment for Nuclear Technology Software (CENTS). CENTS aims to lay the foundation for a Web-based distance computing facility for executing nuclear engineering codes. Through its Web-based interfaces, CENTS will allow users to focus on the problem to be solved instead of the specifics of a particular nuclear code. Via the Web, users will submit input data with computing options for execution, monitor the status of their submissions, retrieve computational results, and use CENTS tools for viewing and analyzing result data.

For computational services, CENTS employs a collection of heterogeneous computer systems logically clustered and managed for optimal resource utilization. The prototype environment was accomplished by integrating the NetSolve system and using Monte Carlo Neutral Particle (MCNP) codes via NetSolve's framework. The user is required only to supply the input problem for the MCNP code. After the user supplies the input, NetSolve sends the problem to the most suitable workstation in the environment; the problem is solved, and the output (4 files) is sent back to the user via the web interface.

CONCLUSION AND FUTURE WORK

We continue to evaluate the NetSolve model to determine how we can architect the system to meet the needs of our users. Our vision is that NetSolve will be used mostly by computational scientists who are not particularly interested in the mathematical algorithms used in the computational solvers, but use them only as means to do research and simulations in their respective domains, whether it is nuclear engineering or computational chemistry. NetSolve will be especially helpful when lots of computational power is needed to do "embarrassingly parallel" tasks. Yet, there lies much room for improvement. We envision future work in features like dynamically extensible servers whose configuration can be modified on-the-fly. The new strategy will be to implement a just-in-time binding of the hardware and software service components, potentially allowing servers to dynamically download software components from service repositories. Parallel libraries could be better supported by data distribution/collection schemes that will marshal input data directly from the client to all computational nodes involved, and collect results in a similar fashion. Efforts also need to be made so that clients can solve jobs with large data sets on parallel machines; the current implementation requires this data to be in-core since the calling sequence expects a reference to the data and not a file pointer, and this may not be possible.

As researchers continue to investigate feasible ways to harness computational resources, the NetSolve system will continue to emerge as a leader in Grid technology. Its light weight and ease of use make it an ideal candidate for middleware, and as we discover the needs of computational scientists, the NetSolve system will be molded to become even more accommodating.

REFERENCES

1. I. Foster and C. Kesselman, editors. *The Grid, Blueprint for a New computing Infrastructure.* Morgan Kaufmann Publishers, Inc., 1998.
2. D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Sagi, Z. Shi, and S. Vahdiyar. Users' Guide to NetSolve V1.4. Technical Report UT-CS-01-467, Computer Science Department, University of Tennessee, July 2001.
3. Inc. Sun Microsystems. XDR: External Data Representation Standard. RFC 1014, Sun Microsystems, Inc., Jun. 1987.
4. B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33–38, September 1994.
5. IBM LoadLeveler for AIX. Technical Report SA22-7311-01, April 2000. http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/sp32/LoadL/lllv2mst.html.
6. PBS Pro Technical Overview. http://www.pbspro.com/tech_overview.html.
7. Inc. The Math Works. *Using MATLAB Version 5.* The Math Works, Inc., 1992.
8. S. Wolfram. *The Mathematica Book, Third Edition.* Wolfram Median, Inc. and Cambridge University Press, 1996.
9. I. Foster and C Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.
10. M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proc. of the 8th International Conference of Distributed Computing Systems, San Jose, CA*, pages 104–111, June 1988.
11. A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Jr. Reynolds. A Synopsis of the Legion Project. Technical Report CS-94-20, Department of Computer Science, University of Virginia, 1994.
12. D. C. Arnold, D. Bachmann, and J. Dongarra. Request Sequencing: Optimizing Communication for the Grid. In A. Bode, T. Ludwig, W. Karl, and R. Wismuller, editors, *Euro-Par 2000 – Parallel Processing*, pages 1213–1222. Springer-Verlag, August 2000.

13. M. Beck and T. Moore. The Internet2 Distributed Storage Infrastructure Project: An Architecture for Internet Content Channels. *Computer Networking and ISDN Systems*, 30(22-23):2141–2148, 1998.
14. J. Plank, M. Beck, W. Elwasif, , T. Moore, M. Swany, and R. Wolski. IBP – The Internet Backplane Protocol: Storage in the Network. In *NetStore '99: Network Storage Symposium, Seatle, WA*, October 1999.
15. H. Casanova, M. Kim, J. S. Plank, and J. Dongarra. Adaptive Scheduling for Task Farming with Grid Middleware. *The International Journal of Supercomputer Applications and High Performance Computing*, 13:231–240, 1999.
16. S. Balay, W. D. Gropp, and B. F. Smith. *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.
17. S. A Hutchinson, Shadid J. N., and Tuminaro R. S. Aztec User's Guide: Version 1.1. Technical Report SAND95-1559, Sandia National Laboratories, 1995.
18. D. C. Arnold, S. Blackford, J. Dongarra, V. Eijkhout, and T. Xu. Seamless Access to Adaptive Solver Algorithms. In *16th IMACS WORLD CONGRESS 2000 on Scientific Computation, Applied Mathematics and Simulation*, August 2000. http://imacs2000.epfl.ch/Abstracts/RechAbstSess.asp, Session: 172, Paper: 6.
19. D. Kincaid, C. Lawson, R. Hanson, and F. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. 5:308–325, 1979.
20. S. Hammarling, J. Dongarra, J. Du Croz, and R. Hanson. An Extended Set of Fortran Basic Linear Algebra Subprograms. 14(1):1–32, 1988.
21. I. Duff, J. Dongarra, J. Du Croz, and S. Hammarling. A Set of Level 3 Basic Linear Algebra Subprograms. 16(1):1–17, 1990.
22. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
23. 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.
24. D. Young, D. Kincaid, J. Respess, and R. Grimes. Itpack 2c: A Fortran Package for Solving Large Sparse Linear Systems by Adaptive Accelerated Iterative Method. Technical report, 1996.
25. I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matices*. Clarendon Press, Oxford, 1986.
26. X. Li. *Sparse Gaussian Elimination on High Performance Computers*. PhD thesis, University of California at Berkeley, 1996. Computer Science Dept.
27. R. Lehoucq, D. Sorensen, and C. Yang. *ARPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, first edition, 1998.
28. R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. Technical Report TR-CS96-494, U.C. San Diego, October 1996.
29. D. C. Arnold, W. Lee, J. Dongarra, and M. Wheeler. Providing Infrastructure and Interface to High-Performance Applications in a Distributed Setting. In Adrian Tentner, editor, *High Performance Computing 2000*, pages 248–253. Society for Computer Simulation International, April 2000.
30. H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proceedings of SuperComputing 2000 (SC'00)*, Nov. 2000.