# NetSolve: Grid Enabling Scientific Computing Environments

Keith Seymour, Asim YarKhan, Sudesh Agrawal, and Jack Dongarra

*Computer Science Department, University of Tennessee, Knoxville, TN 37919 USA*

**Abstract**

The purpose of NetSolve is to create the middleware necessary to provide a seamless bridge between the simple, standard programming interfaces and desktop systems that dominate the work of computational scientists and the rich supply of services supported by the emerging Grid architecture, so that the users of the former can easily access and reap the benefits (shared processing, storage, software, data resources, etc.) of using the latter.

## 1 Introduction

Researchers today are becoming increasingly reliant on the availability of computational resources to assist in their research and quantify their findings. The creation of Grids has helped increase the availability of such resources by combining the computing power and storage resources of many computers into one. However, harnessing these combined resources is still a challenge and out of reach for many scientists. As a mechanism for providing access to various distributed hardware and software resources, NetSolve reduces the complexity of locating, accessing, and using the computational resources necessary for conducting timely research. To accomplish this goal, NetSolve provides the middleware necessary to provide a seamless bridge between the simple, standard programming interfaces and desktop systems that dominate the work of computational scientists and the rich supply of services supported by the emerging Grid architecture, so that the users of the former can easily access and reap the benefits (shared processing, storage, software, data resources, etc.) of using the latter. This vision of the broad community of scientists, engineers, research professionals and students, working with the powerful and flexible tool set provided by their familiar desktop computing environment, and yet able to easily draw on the vast, shared resources of the Grid for unique or exceptional resource needs, or to collaborate intensively with colleagues in other organizations and locations, is the vision that NetSolve is designed to realize.

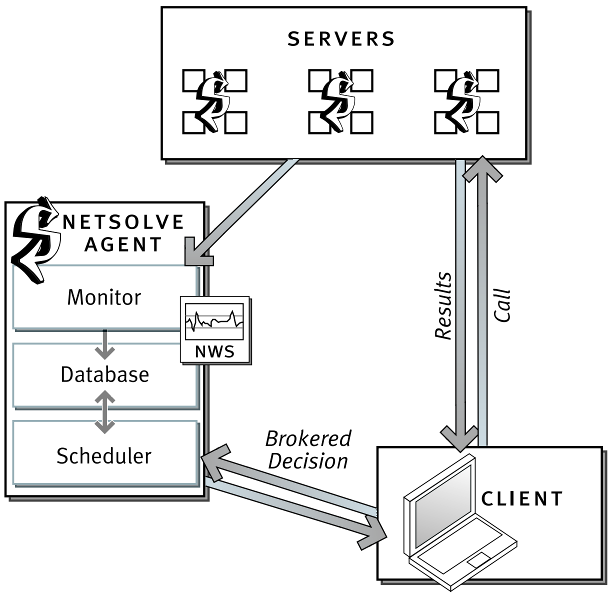


Fig. 1. Overview of NetSolve

## 2 Overview of NetSolve

NetSolve is a client-agent-server system which provides remote access to hardware and software resources from a variety of scientific problem solving environments such as Matlab, Mathematica, and Octave, as well as traditional programming languages like C and Fortran. In this section, we provide an overview of the architecture and features of NetSolve.

### *2.1 NetSolve Architecture*

A NetSolve system consists of three entities, as illustrated in Figure 1.

- The *Client*, which needs to execute some remote procedure call. As we mentioned, the client may be invoked from an interactive problem solving environment or by a standalone program written in a traditional programming language.
- The *Server* executes functions on behalf of the clients. The server hardware can range in complexity from a uniprocessor to a MPP system and the functions executed by the server can be arbitrarily complex. Server administrators can straightforwardly add their own services without affecting the rest of the NetSolve system.
- The *Agent* is the focal point of the NetSolve system. It maintains a list of all available servers and performs resource selection for client requests as well as ensuring load balancing of the servers.

In practice, from the user's perspective the mechanisms employed by NetSolve make the remote procedure call fairly transparent. However, behind the scenes, a typical call to NetSolve involves several steps, as follows:

(1) The client queries the agent for an appropriate server that can execute the desired function.
(2) The agent returns a list of available servers, ranked in order of suitability.
(3) The client attempts to contact a server from the list, starting with the first and moving down through the list. The client then sends the input data to the server.
(4) Finally the server executes the function on behalf of the client and returns the results.

In addition to providing the middleware necessary to perform the brokered remote procedure call, NetSolve aims to provide mechanisms to interface with other existing Grid services. This can be done by having a client that knows how to communicate with various Grid services or by having servers that acts as a proxies to those Grid services. NetSolve provides some support for the proxy server approach, while the client-side approach would be supported by the emerging GridRPC standard API [1]. We briefly discuss these two approaches here.

Normally the NetSolve server executes the actual service request itself, but in some cases it can act as a proxy to specialty mechanisms such as Condor, MPI, and ScaLAPACK. The primary benefit is that the client-to-server communication protocol is identical so the client does not need to be aware of every possible back-end service. A server proxy also allows aggregation and scheduling of resources, such as the machines in a cluster, on one NetSolve server.

The GridRPC API represents ongoing work to standardize and implement a portable and simple remote procedure call (RPC) mechanism for grid computing. This standardization effort is being pursued through the Global Grid Forum Research Group on Programming Models [2]. The initial work on GridRPC reported in [1] shows that client access to existing grid computing systems such as NetSolve and Ninf [3] can be unified via a common API, a task that has proved to be problematic in the past. In its current form, the C API provided by GridRPC allows the source code of client programs to be compatible with different Grid services, provided that service implements a GridRPC API. We describe the capabilities of the GridRPC API in Section 3.

The combination of these technologies will allow NetSolve to provide seamless client access to a diverse set of Grid services.

## *2.2 Strengths of NetSolve*

### *2.2.1 Ease of Use*

The primary design goal of NetSolve is to make it easy to access grid resources, even for users who do not have much expertise in programming. We have accomplished this by providing interfaces to interactive scientific computing environments (Matlab, Octave, and Mathematica) which themselves are easy to use and place a low burden on the casual user. This design philosophy extends to the interfaces for traditional programming languages, which match the original function calls as closely as possible, thus reducing the amount of effort required to convert software to use NetSolve. For example, a direct call from Fortran77 to the LAPACK routine `DGESV` looks like:

```
CALL DGESV( N, 1, A, LDA, IPIV, B, LDB, INFO )
```

The equivalent call using NetSolve is:

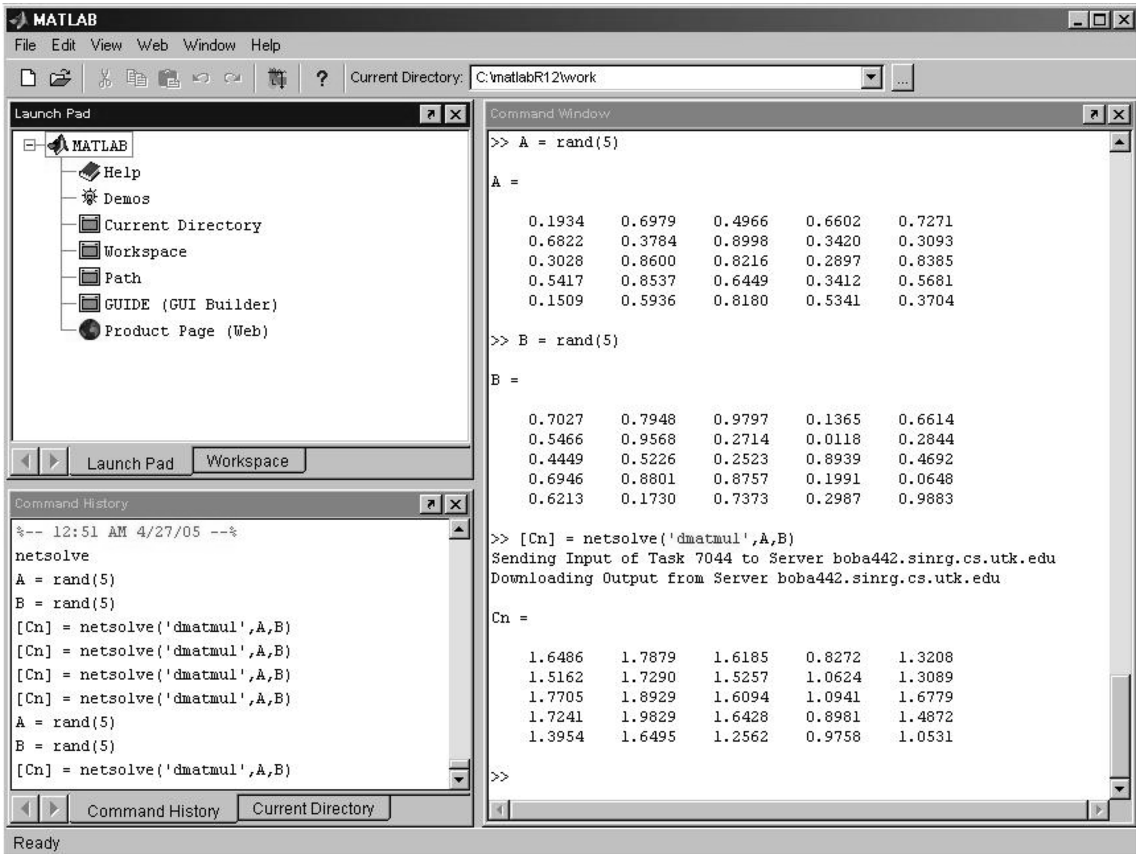
```
CALL FNETSL( 'DGESV()', STATUS,
            N, 1, A, LDA, IPIV, B, LDB, INFO )
```

In addition, because NetSolve dynamically determines the calling sequence, no stub code needs to be linked with the client. In typical RPC systems, there is a formal description of the calling sequence of the procedure which is used to generate stub code. On the client side, the user calls the stub instead of the actual procedure. The client stub handles marshalling the arguments to and from the remote server. Since the client stub must be compiled and linked with the user's code before it can be called, calling dynamically installed procedures from interactive environments is difficult. In contrast, the NetSolve client library determines the calling sequence at run-time by downloading a specification from the server. This specification is used to determine how to marshall the arguments to and from the server. Therefore, the user never needs to link stub code with their application, making it straightforward to use NetSolve from within interactive environments.

An example of the ease of using NetSolve from Matlab is shown in Figure 2. In this case, we generate two matrices using the Matlab random number generator and pass them to NetSolve to be multiplied using the `dmatmul` service. The result is assigned back to a Matlab variable and can be used in subsequent operations as usual.

### *2.2.2 Pre-configured Numerical Routines*

In order to make the system immediately useful, NetSolve comes with pre-configured interfaces to a variety of numerical routines including dense linear algebra (from



```
%-- 12:51 AM 4/27/05 --%
netsolve
A = rand(5)
B = rand(5)
[Cn] = netsolve('dmatmul',A,B)
[Cn] = netsolve('dmatmul',A,B)
[Cn] = netsolve('dmatmul',A,B)
[Cn] = netsolve('dmatmul',A,B)
A = rand(5)
B = rand(5)
[Cn] = netsolve('dmatmul',A,B)
```

```
>> A = rand(5)

A =

    0.1934    0.6979    0.4966    0.6602    0.7271
    0.6822    0.3784    0.8998    0.3420    0.3093
    0.3028    0.8600    0.8216    0.2897    0.8385
    0.5417    0.8537    0.6449    0.3412    0.5681
    0.1509    0.5936    0.8180    0.5341    0.3704

>> B = rand(5)

B =

    0.7027    0.7948    0.9797    0.1365    0.6614
    0.5466    0.9568    0.2714    0.0118    0.2844
    0.4449    0.5226    0.2523    0.8939    0.4692
    0.6946    0.8801    0.8757    0.1991    0.0648
    0.6213    0.1730    0.7373    0.2987    0.9883

>> [Cn] = netsolve('dmatmul',A,B)
Sending Input of Task 7044 to Server boba442.sinrg.cs.utk.edu
Downloading Output from Server boba442.sinrg.cs.utk.edu

Cn =

    1.6486    1.7879    1.6185    0.8272    1.3208
    1.5162    1.7290    1.5257    1.0624    1.3089
    1.7705    1.8929    1.6094    1.0941    1.6779
    1.7241    1.9829    1.6428    0.8981    1.4872
    1.3954    1.6495    1.2562    0.9758    1.0531

>>
```

Fig. 2. Example of Using NetSolve from Matlab

BLAS and LAPACK), sparse linear algebra, parallel solvers, and other miscellaneous examples. These problems can be selectively enabled by the server administrator depending on which libraries are available on the machine. In an attempt to provide a useful service to users who cannot start their own servers, the publicly available NetSolve grid running at the Univeristy of Tennessee has all the problems enabled. These pre-configured interfaces also serve as examples of how to integrate external software into NetSolve.

### *2.2.3 Agent as Resource Broker*

One of the difficulties inherent in Grid computing is locating the best resources available to execute the requested service. In agent based scheduling, the NetSolve agent uses knowledge of the requested service, information about the parameters of the service request from the client, and the current state of the resources to score the possible servers and return the servers in sorted order.

When a service is started, the server informs the agent about services that it provides and the computational complexity of those services. This complexity is expressed using two integer constants $a$ and $b$ and is evaluated as $aN^b$, where $N$ is the size of the problem. At startup, the server notifies the agent about its computational speed (approximate MFlops from a simple benchmark) and it continually updates the agent with information about its workload. The bandwidth and latency of communication between the server and the agent are also monitored, and are used as an estimate of the communication capacity between the client and server. When an agent receives a request for a service with a particular problem size, it uses the problem complexity and the server status information to estimate the time to completion on each server providing that service. It orders the servers in terms of fastest time to completion, and then returns the list of servers to the client. This scheduling heuristic is known as *Minimum Completion Time* and it works well in many practical cases. Each service request would be assigned to the server that would complete the service in the minimum time, assuming that the currently known loads on the servers will remain constant during the execution.

### *2.2.4 Resource Aggregation*

Another of NetSolve's goals is to provide a middleware infrastructure that can be used to simplify access to complicated software, such as parallel or distributed programs. As we introduced in Section 2.1, in this approach NetSolve creates server-proxies to delegate the client requests to external scheduling and execution services such as batch systems, Condor, ScaLAPACK, or LFC (LAPACK for Clusters). The other NetSolve components see the server-proxy as a single entity, even though it can represent a large number of actual resources.

The key to this approach lies in the way services are defined. Each NetSolve service is defined by a formal description of the calling sequence, including the arguments, data types, and dimensions. This is generally referred to as an Interface Definition Language. Based on this description, both the client and server know how to marshall and unmarshall the input and output data. The actual implementation of the service is contained in an external library and is linked in with the service at compilation time. Therefore, any library with the same calling sequence can be linked using the same IDL, leaving the communication protocol unchanged. This allows having multiple NetSolve servers that implement the same service in vastly different ways. For example, a matrix multiplication service could be implemented on one machine using a vendor-tuned linear algebra library and on another machine using the reference BLAS [4]. Because the NetSolve communication protocol is the same for both, the client does not need to be concerned with which server is selected for the request.

This server-proxy approach has been used to provide simple access to parallel MPI jobs, as well as aggregate resources managed by LSF [5], Condor-G [6], and LFC [7]. In this way, we can bring the power of these aggregate resources to the user's familiar Matlab or Mathematica environment while leaving the server administrator free to optimize or modify the implementation without concern for the client.

### *2.2.5 Transparent Fault Tolerance*

There are a variety of reasons a server may be unable to satisfy a client's request. In some cases, it is caused by policy decisions rather than technical problems. For instance, the server may have already accepted the maximum number of jobs the administrator configured it to handle or the server may be configured to require authentication and the client does not have the proper credentials. In those cases, the server is operating normally, but policy requires rejecting those requests. In other cases, however, the problem may be caused by genuine technical problems, such as the machine going down, the NetSolve server being stopped, network connectivity problems, or a poorly implemented service that crashes before the solution is computed.

The NetSolve approach to handling these problems is to automatically resubmit the job to another server. When the client initially submits a request to the agent, rather than returning a single server, it returns a list of servers. The client starts with the first server on the list and if a failure is encountered, it resubmits to the next server on the list. This continues until there are no servers remaining. The process is completely transparent to the user, so they only see a call fail if all servers that implement that service have failed.

### *2.2.6 Request Sequencing*

As the size of data sets increases, the ability to specify the flow of data becomes more important. It would be inefficient to force intermediate results to be transmitted back and forth between the client and servers when those results will not be used again on the client and are needed at the server during the future steps of the computation. Our aim in *request sequencing* is to decrease network traffic between client and server components in order to decrease overall request response time. Our design ensures that (i) no unnecessary data is transmitted and (ii) all necessary data is transferred. This is accomplished by performing a data flow analysis of the input and output parameters of every request in the sequence to produce a directed acyclic graph (DAG) that represents the tasks and their execution dependences. This DAG is then sent to a server in the system where it is scheduled for execution.

In the current version of request sequencing, the NetSolve agent assigns the entire sequence to a single server. The server is selected based on the sum of the predicted run times of all the tasks. We execute a node if all its inputs are available and there are no conflicts with its output parameters. Because the only mode of execution we currently support is using a single NetSolve server, NetSolve is prevented from exploiting any parallelism inherent in the task graph. However, distributing independent tasks to different machines makes the scheduling and data management more important (and complicated) than in the single server scenario. This limitation would also be a problem when no single server has all the software required to execute the entire sequence. Therefore, we will need to extend the NetSolve scheduling and execution infrastructure to make more efficient use of the available resources when executing a sequenced request.

As a simple demonstration to show the effectiveness of request sequencing, we submit a sequence of three NetSolve requests, each dependent on the results of the previous request: `DGEMM` ⇒ `DLACPY` ⇒ `DGEMM`. `DGEMM` is a matrix multiply routine and `DLACPY` copies a matrix. Because of the sequential nature of this task graph, even when multiple servers are available, the tasks cannot execute in parallel. Submitting this sequence to a dual processor 933MHz Pentium 3 machine as three individual requests took 4.43 seconds on average, while submitting it as one sequence only took 3.07 seconds because some data movement was avoided. Of course, if the task graph contained some inherent parallelism, using a non-sequenced series of requests would allow us to execute on multiple machines, possibly closing the performance gap. However, the difference in performance depends on the amount of parallelism that we could exploit and the communication overhead that we would avoid by using request sequencing.

### *2.2.7 Task Farming in NetSolve*

*Task Farming* represents an important class of distributed computing applications, where multiple independent tasks are executed to solve a particular problem. Many algorithms fit into this framework, for example, parameter-space searches, Monte-Carlo simulations and genome sequence matching. This class of applications is highly suited to Grid computing, however, scheduling task farming applications efficiently can be difficult since the resources in a Grid may be highly variable and the tasks may take different amounts of time.

Without using a special task farming API, a naïve algorithm could be implemented by using the standard NetSolve interface and letting the NetSolve agent handle the scheduling. A user would make a series of non-blocking requests, probe to see if the requests have completed, and then wait to retrieve the results from completed requests. However this leads to problems with regard to scheduling, especially if the number of tasks is much larger than the number of servers. Alternatively, the user could try to handle the details of scheduling, but this solution requires a knowledge of the system that is not easily available to the user, and it ignores the NetSolve goal of ease-of-use.

In order to provide an efficient and easy to use interface to task farming, NetSolve implements a special API. In the farming interface, the user converts the parameters for requests into an arrays of parameters, indexed by an iterator string. Figure 3 shows an example of the task farming interface. The task farming API only adds 4 calls to NetSolve, namely 3 calls for constructing arrays of different data types, and

Using standard non-blocking NetSolve API

```
requests1 = netslnb('iqsort()',size1, ptr1,sort1);
requests2 = netslnb('iqsort()',size2, ptr2,sort2);
...
requests200 = netslnb('iqsort()',size200, ptr200,sorted200);
for each request probe for completion with netslpr()
for each request wait for results using netslwt()
```

Using task farming API

```
int sizearray[200];
void *ptrarray[200];
void *sortedarray[200];
sizearray[0] = size1;
ptrarray[0] = ptr1;
sortedarray[0] = sorted1;
...
statusarray = netsl_farm("i=0,199","iqsort()",
ns_int_array(sizearray,"$i"), ns_ptr_array(ptrarray,"$i"),
ns_ptr_array(sortedarray,"$i"));
```

Fig. 3. Task Farming Example: A integer quicksort routine is implemented using standard non-blocking calls (top) and then converted to using the task farming interface (bottom).

1 call for the actual farming. More details about the API can be found in the Users Guide to NetSolve [8].

One problem with the current task farming API is that it only returns when the all the tasks have been completed. That is, it does not allow the user to get results when a subset of the tasks have been completed, so the user cannot visualize, guide or cancel during the execution. These are things that we are working to address in the current development version of NetSolve (known as GridSolve).

### *2.2.8 Distributed Storage Infrastructure*

As the size of the data sets used in scientific computing continues to increase, storage management becomes more important for grid applications. In the normal NetSolve RPC model, the client has all the input data resident in memory or in a local file. The client sends the data directly to the server and retrieves the results directly from the same server. Of course, the main limitation of this model is that the data set cannot exceed the storage capacity of the client, which could be a very severe restriction. In these cases, some form of remote storage management will be essential.

The Distributed Storage Infrastructure (DSI) is an attempt towards achieving coscheduling of the computation and data movement over the NetSolve Grid. The DSI API helps the user in controlling the placement of data that will be accessed by a Net-Solve service. This is useful in situations where a given service accesses a single block of data a number of times. Instead of multiple transmissions of the same data from the client to the server, the DSI feature helps to transfer the data from the client to a storage server just once, and relatively cheap multiple transmissions from the storage server to the computational server. Thus the present DSI feature helps NetSolve to operate in a cache-like manner. Presently, only the Internet Backplane Protocol (IBP) [9] is used for providing the storage service. IBP provides mid-dleware for managing and using remote storage, but we hide the specifics of the IBP interface behind the DSI layer. This will allow us to integrate other commonly available storage service systems in the future without requiring changes to client code already written using DSI.

## 3 GridRPC API

GridRPC is a standardized, portable, and simple programming interface for remote procedure call (RPC) over the Grid. NetSolve provides a GridRPC layer on top of the normal API, but in the future, GridRPC will be the primary end-user API. In this section, we informally describe the GridRPC model and the functions that comprise the API. The current Global Grid Forum Recommendation Draft for GridRPC [10] contains a detailed listing of the function prototypes, however the version currently implemented by NetSolve is described in detail in [1].

### *3.1 Function Handles and Session IDs*

Two fundamental objects in the GridRPC model are *function handles* and *session IDs*. The function handle represents a mapping from a function name to an in-stance of that function on a particular server. The GridRPC API does not dictate the mechanics of resource discovery since different underlying GridRPC imple-mentations may use vastly different protocols. Once a particular function-to-server mapping has been established by initializing a function handle, all RPC calls us-ing that function handle will be executed on the server specified in that binding. A session ID is an identifier representing a particular non-blocking RPC call. The session ID is used throughout the API to allow users to obtain the status of a pre-viously submitted non-blocking call, to wait for a call to complete, to cancel a call, or to check the error code of a call.

### *3.2 Initializing and Finalizing Functions*

The initialize and finalize functions are similar to the MPI initialize and finalize calls. Client GridRPC calls before initialization or after finalization will fail.

- `grpc_initialize` reads the configuration file and initializes the required modules.
- `grpc_finalize` releases any resources being used by GridRPC.

### *3.3 Remote Function Handle Management Functions*

The *function handle management* group of functions allows creating and destroying function handles.

- `grpc_function_handle_default` creates a new function handle using a default server. This could be a pre-determined server name or it could be a server that is dynamically chosen by the resource discovery mechanisms of the underlying GridRPC implementation, such as the NetSolve agent.
- `grpc_function_handle_init` creates a new function handle with a server explicitly specified by the user.
- `grpc_function_handle_destruct` releases the memory associated with the specified function handle.
- `grpc_get_handle` returns the function handle corresponding to the given session ID (that is, corresponding to that particular non-blocking request).

### *3.4 GridRPC Call Functions*

The four GridRPC call functions may be categorized by a combination of two properties: blocking behavior and calling sequence. A call may be either blocking (synchronous) or non-blocking (asynchronous) and it may use either a variable number of arguments (like `printf`) or an *argument stack* calling sequence. The argument stack calling sequence allows building the list of arguments to the function at runtime through elementary stack operations, such as *push* and *pop*.

- `grpc_call` makes a blocking remote procedure call with a variable number of arguments.
- `grpc_call_async` makes a non-blocking remote procedure call with a variable number of arguments.
- `grpc_call_argstack` makes a blocking call using the argument stack.
- `grpc_call_argstack_async` makes a non-blocking call using the argument stack.

## *3.5 Asynchronous GridRPC Control Functions*

The following functions apply only to previously submitted non-blocking requests.

- `grpc_probe` checks whether the asynchronous GridRPC call has completed.
- `grpc_cancel` cancels the specified asynchronous GridRPC call.

## *3.6 Asynchronous GridRPC Wait Functions*

The following five functions apply only to previously submitted non-blocking requests. These calls allow an application to express desired non-deterministic completion semantics to the underlying system, rather than repeatedly polling on a set of sessions IDs. (From an implementation standpoint, such information could be conveyed to the OS scheduler to reduce cycles wasted on polling.)

- `grpc_wait` blocks until the specified non-blocking requests to complete.
- `grpc_wait_and` blocks until *all* of the specified non-blocking requests in a given set have completed.
- `grpc_wait_or` blocks until *any* of the specified non-blocking requests in a given set has completed.
- `grpc_wait_all` blocks until *all* previously issued non-blocking requests have completed.
- `grpc_wait_any` blocks until *any* previously issued non-blocking request has completed.

## *3.7 Error Reporting Functions*

Of course it is possible that some GridRPC calls can fail, so we need to provide the ability to check the error code of previously submitted requests. The following error reporting functions provide error codes and human-readable error descriptions.

- `grpc_get_last_error` returns the error code for the last invoked GridRPC call.
- `grpc_get_error` returns the error code associated with a given non-blocking request.
- `grpc_perror` prints the error string associated with the last GridRPC call.
- `grpc_error_string` returns the error description string, given a numeric error code.

### *3.8 Argument Stack Functions*

When describing the GridRPC call functions, we mentioned that there is an alternate calling style that uses an *argument stack*. With the following functions it is possible to construct the arguments to a function call at run-time. When interpreted as a list of arguments, the stack is ordered from bottom up. That is, to emulate a function call `f(a,b,c)`, the user would push the arguments in the same order: `push(a); push(b); push(c);`.

- `grpc_arg_stack_new` creates a new argument stack.
- `grpc_arg_stack_push_arg` pushes the specified argument onto the stack.
- `grpc_arg_stack_pop_arg` removes the top element from the stack.
- `grpc_arg_stack_destruct` frees the memory associated with the specified argument stack.

## 4 GridSolve: The Future of NetSolve

Over time, many enhancements have been made to NetSolve to extend its functionality or to address various limitations. Task farming, request sequencing, and security are examples of enhancements made after the original implementation of NetSolve. However, some desirable enhancements cannot be easily implemented within the current NetSolve framework. Thus, our ongoing work on NetSolve involves redesigning the framework from the ground up to address some of these new requirements.

Based on our experience developing NetSolve we have identified several requirements that are not adequately addressed in the current NetSolve system. These new requirements - coupled with the requirements for the original NetSolve system - will form the basis for our next generation middleware, known as GridSolve.

The overall goal is to address three general problems: server-side ease of use, interoperability, and scalability. Improving server-side ease of use primarily refers to improving the process of integrating external code and libraries into a GridSolve server. Interoperability encompasses several facets, including better handling of different network topologies (such as those including NATs), better support for parallel libraries and parallel architectures, and better interaction with other Grid computing systems such as Globus [11] and Ninf [3]. Scalability in the context used here means that system performance does not degrade as a result of adding components or increasing the number of requested services in the GridSolve system.

This section describes some of the specific solutions to the general problems discussed above.

## *4.1 Network Address Translators*

As the rapid growth of the Internet began depleting the supply of IP addresses, it became evident that some immediate action would be required to avoid complete IP address depletion. The IP Network Address Translator [12] is a short-term solution to this problem. Network Address Translation presents the same external IP address for all machines within a private subnet, allowing reuse of the same IP addresses on different subnets, thus reducing the overall need for unique IP addresses.

As beneficial as NATs may be in alleviating the demand for IP addresses, they pose many significant problems to developers of distributed applications such as GridSolve [13]. Some of the problems as they pertain to GridSolve are: IP addresses may not be unique, IP address-to-host bindings may not be stable, hosts behind the NAT may not be contactable from outside, and NATs may increase network failures.

To address these issues we have developed a new communications framework for GridSolve. To avoid problems related to potential duplication of IP addresses, the GridSolve components will be identified by a globally unique identifier specified by the user or generated randomly. In a sense, the component identifier is a network address that is layered on top of the real network address such that a component identifier is sufficient to uniquely identify and locate any GridSolve component, even if the real network addresses are not unique. This is somewhat similar to a machine having an IP address layered on top of its MAC address in that the protocol to obtain the MAC address corresponding to a given IP address is abstracted in a lower layer. Since NATs may introduce more frequent network failures, we have implemented a mechanism that allows a client to submit a problem, break the connection, and reconnect later at a more convenient time to retrieve the results.

An important aspect to making this new communications model work is the *GridSolve proxy*, which is a component that allows servers to exist behind a NAT. Since a server cannot easily accept unsolicited connections from outside the private network, it must first register with a proxy. The proxy acts on behalf of the component behind the NAT by establishing connections with other components or by accepting incoming connections. The component behind the NAT keeps the connection with the proxy open as long as possible since it can only be contacted by other components while it has a control connection established with the proxy. To maintain good performance, the proxy only examines the header of the connections that it forwards and it uses a simple table-based lookup to determine where to forward each connection. Furthermore, to prevent the proxy from being abused, authentication may be required.

## *4.2 Scheduling Enhancements*

GridSolve will retain the familiar agent-based and server-based scheduling of resources, but in many cases the client has the most accurate knowledge about how to select the best resource. Therefore we are implementing an infrastructure that allows filtering and scheduling to be optionally performed by the client.

In the current NetSolve system, the only filter that affects the selection of resources is the problem name. Given the problem name, the NetSolve agent filters to select the servers that can solve that problem, then chooses the “best” server. However, the notion of which server is best is entirely determined by the agent. In GridSolve, we are extending this behavior. We allow the user to provide constraints on the filtering and selection process. These selection constraints imply that the user has some knowledge of which characteristics will lead to a better solution to the problem (most likely in terms of speed), for example, a minimum memory requirement. Also we will allow the user to have access to the complete list of resources and their characteristics so that the client can implement comprehensive scheduling algorithms in addition to simple filtering.

To make this functionality useful, the GridSolve servers should provide as much information as possible to the agent, in turn providing a flexible environment to the client for its request. To make the best selection for the client, the agent uses this information stored in the form of resource attributes and performs the filtering on behalf of the client. Furthermore, we allow the service providers (that is, those organizations that provide GridSolve servers) to specify constraints on the clients that can access that service. For example, an organization may want to restrict access to a certain group of collaborators. This information is also specified in the resource attributes of the service.

Since the GridSolve agent currently maintains information about all resources in the entire system, it can be viewed as the main performance bottleneck as more resources are added. The natural approach to this problem is to use multiple agents such that the load on each agent is reduced. However, this distributed approach leads to some interesting scheduling issues since each agent might only store information about its local domain. While each agent may prefer to schedule jobs within its domain, it may actually be more efficient to send the job to another agent if the computational and network communication requirements warrant. Thus, some agent-to-agent communication will certainly be required when using multiple agents.

## *4.3 IDL Improvements*

One of the original design goals of NetSolve was to eliminate the need for client-side stubs for each procedure in a remote procedure call (RPC) environment. However, this design decision tends to push the complexity to the servers. Integrating new software into NetSolve required writing a complex server side interface definition (Problem Description File), which specifies the parameters, data types, and calling sequence. Despite several attempts to create a user-friendly tool to generate the Problem Description Files, it can still be a difficult and error-prone process.

Therefore, we have implemented a simple technique for adding additional services to a running server. The interface definition format itself has been greatly simplified and the services are compiled as external executables with interfaces to the server described in a standard format. The server re-examines its own configuration and installed services periodically or when it receives the appropriate signal. In this way it becomes aware of any additional services that are installed without re-compilation or restarting.

Integrating parallel software has been difficult in some cases because the Problem Description File format does not support it in a general way. Additionally, some parallel software has required using a customized server. Making parallel software easier to integrate into GridSolve hinges on two issues: the server should support it in a general way and the interface definition language should be extended to allow specifying additional parameters, such as the number of processors to be used. We are continuing to work on these issues.

## 5 Applications using NetSolve

The desire to make software available through a computational grid may be motivated by several factors. In some cases, the goal is to parallelize the computation to reduce the total time to solution. Grid middleware can also help increase overall resource utilization by balancing the load among the available servers. In other cases the goal is simply to provide remote access to software that is not feasible to run on the user's machine, perhaps because of high resource requirements, dependence on specialized libraries, or source code portability. In this section, we describe a diverse set of applications that have been grid-enabled using NetSolve.

### *5.1 Environmental Modeling*

A tremendous amount of planning goes into an undertaking as large as restoring the Everglades. Studies must first be done to determine what areas need to be restored and how best to do so without further damaging an already delicate ecosystem. To aid in this planning, a group at the University of Tennessee led by Dr. Lou Gross has collaborated on the development of a suite of environmental models called ATLSS (Across Tropic Level System Simulation) [14]. These models provide comparisons of the effects of alternative future hydrologic plans on various components of the biota.

This package has proven itself quite useful in the planning efforts, however it requires extensive computational facilities that are typically not available to the many stakeholders (including several federal and state agencies) involved in the evaluation of plans for restoration that are estimated to cost $8 billion. To allow greater access and use of computational models in the South Florida stakeholder community, a grid-enabled interface to the ATLSS models has been developed and is currently being used on SInRG resources. This interface provides for the distribution of model runs to heterogeneous grid nodes. The interface utilizes NetSolve for model run management and the LoRs (Logistical Runtime System) [15] toolkit and library for data and file movement. Integration of the grid interface with a web based launcher and database provides a single interface for accessing, running, and retrieving data from the variety of different models that make up ATLSS, as well as from a variety of different planning scenarios.

ATLSS, in conjunction with NetSolve and LoRS, is the first package we are aware of that provides transparent access for natural resource managers through a computational grid to state-of-the-art models. The interface allows users to choose particular models and parameterize them as the stakeholder deems appropriate, thus allowing them the flexibility to focus the models on particular species, conditions or spatial domains they wish to emphasize. The results can then be viewed within a separate GIS tool developed for this purpose.

## *5.2 Statistical Parametric Mapping*

Statistical Parametric Mapping (SPM) is a widely used medical imaging software package. The SPM web site [16] describes the technique as follows.

> Statistical Parametric Mapping refers to the construction and assessment of spatially extended statistical process used to test hypotheses about [neuro]imaging data from SPECT/PET & fMRI. These ideas have been instantiated in software that is called SPM.

Although SPM has achieved widespread usage, little work has been done to optimize the package for better performance. In particular, little effort has gone into taking advantage of the largely parallel nature of many parts of the SPM package.

Through continuing research by Dr. Jens Gregor and Dr. Michael Thomason at the University of Tennessee, preliminary work has been done to enhance the SPM package to utilize grid resources available through NetSolve and IBP by way of the NetSolve-to-IBP library. NetSolve-to-IBP is a library built on top of LoRS and ROLFS (Read-Only Logistical File System) that allows the sharing of files between the NetSolve client and server processes, using IBP repositories as intermediate storage. This allows programs that need access to a common set of files (e.g. SPM) to export some of their functionality to a NetSolve server without having to use a shared filesystem, such as NFS.

The grid-enabled version of SPM is still under development, but executions of the preliminary version have been timed to run in one half to one third the time of unmodified code in some simulations. Once completed, the SPM software will be distributed across the SInRG resources for use in medical research, providing doctors and researchers a faster time to completion, something often critical in medical imaging analysis.

### *5.3 Image Compression*

An image compression application using singular value decomposition was built to demonstrate how the C# interface can be used from within .NET. Singular Value Decomposition (SVD) [17] is a useful mathematical tool for finding and removing information stored in matrix form based on its significance to the rest of the data. Image compression is one of many uses of SVD.

Images are represented as matrices with values in the elements to describe the intensity of the color. Color images are actually a composite of matrices representing different colors; generally red, green, and blue. When the image is decomposed into the form $UDV^T$ by SVD, the singular values are representative of the clarity of the image. When some of the values are discarded the image loses clarity, but this loss in precision is made up for by the reduction in space needed to store the image.

Currently, the prototype NetSolve web service implements `dgesvd` (a singular value decomposition routine from the LAPACK [18] mathematical library) in a non-blocking fashion. Invoking the `dgesvd` web method creates a new thread and immediately returns. The `dgesvd` call ships the image as a .NET DIME [19] attachment to the web service side. The web service then decomposes the image into three matrices representing red, green, and blue. Each of the matrices is then submitted to NetSolve using the `dgesvd` problem and solved. Image reconstruction is done as soon as the client receives enough information from the web service to reconstruct the requested image(s).

### *5.4 Vertex Cover and Clique Problems*

A widely-known and studied problem in computer science and other disciplines is the Vertex Cover problem, which asks the following question.

> Given a graph G=(V,E) and an integer k, does G contain a set S with k or fewer vertices that covers all of the edges in G, where an edge is said to be covered if at least one of its endpoints are contained in S?

Vertex Cover is NP-complete in general, but solvable in polynomial time when k is fixed. The applications for this problem are far-reaching, including applications in bioinformatics, such as phylogeny, motif discovery, and DNA microarray analysis. The problem, however, is inherently difficult and time-consuming to solve, so efficient software packages for solving Vertex Cover are very desirable.

Research conducted by Dr. Michael Langston of the University of Tennessee aims to create an efficient software package for solving Vertex Cover. Dr. Langston and his student researchers are interested mainly in the duality between the Vertex Cover problem and the Clique problem. The Clique problem asks the following question.

> Given a graph G=(V,E) and an integer k, does G contain a set S of k nodes such that there is an edge between every two nodes in the clique?

By exploiting the duality between these two problems, they have been able to solve extremely large instances of Clique (graphs containing greater than 104 vertices). To achieve reasonable times to solution, Dr. Langston's team has developed a parallel version of their software, which is being actively run on SInRG (Scalable Intracampus Research Grid) [20] resources. The team has taken several approaches to making their application grid-aware, ranging from developing a custom scheduler and starting jobs via Secure Shell (SSH) to using popular grid middleware, such as Condor. The team has implemented a prototype version of their software that uses NetSolve to efficiently access a large number of computational resources.

## *5.5 Genetic Algorithms*

Because of their durability and fuel efficiency, diesel engines are installed in many kinds of vehicles ranging from compact cars to large trucks. With increasing environmental concerns and legislated emission standards, current research is focused on reduction of soot and NOx simultaneously while maintaining reasonable fuel economy. In this research, the optimization system designs a diesel engine with small amounts of Soot and NOx along with high fuel efficiency [21]. There are three components: the phenomenological diesel engine model, the Genetic Algorithm, and NetSolve.

HIDECS (Hiroyasu Diesel Engine Combustion Simulation) [22] is the most sophisticated phenomenological spray-combustion model currently available, originally developed at the University of Hiroshima. It has already demonstrated potential as a predictive tool for both performance and emissions in several types of direct injection diesel engines. Genetic Algorithm (GA) is an optimization algorithm that imitates the evolution of living creatures. In nature, creatures inadaptable to an environment meet extinction, and only adapted creatures can survive and reproduce. A repetition of this natural selection spreads the superior genes to conspecifics and then the species prospers. GA models this process of nature on computers.

GA can be applied to several types of optimization problems by encoding design variables of individuals. Searching for the solution proceeds by performing three genetic operations on the individuals: selection, crossover, and mutation, which play an important role in GA. Selection is an operation that imitates the survival of the fittest in nature. The individuals are selected for the next generation according to their fitness. Crossover is an operation that imitates the reproduction of living creatures. The crossover operation exchanges the information of the chromosomes among individuals. Mutation is an operation that imitates the failure that occurs when copying the information of DNA. Mutating the individuals with a proper probability maintains the diversity of the population.

Since it takes a lot of time to derive the optimum solution by GA, parallel processing is preferred. Fortunately GA is a very suitable algorithm for performing parallel processing and the *farming* function of NetSolve is very easy to apply to GA since it provides an easy way to submit a large number of requests to be executed in parallel. In this system, GA is performed on the client side. When the searching points are evaluated, the data is sent to the server and calculated using the faming function. A huge computational cost is required to derive these results. However, NetSolve farming helps this system to reduce the total calculation time.

## 6 Related Work

Several Network Enabled Servers (NES) provide mechanisms for transparent access to remote resources and software. Ninf-G [23] is a reference implementation of the GridRPC API [1] built on top of the Globus Toolkit. Ninf-G provides an interface definition language that allows services to be easily added, and client bindings are available in C and Java. Security, scheduling and resource management are left up to Globus.

The DIET (Distributed Interactive Engineering Toolbox) project [24] is a client-agent-server RPC architecture which uses the GridRPC API as its primary interface. A CORBA Naming Service handles the resource registration and lookup, and a hierarchy of agents handles the scheduling of services on the resources. An API is provided for generating service profiles and adding new services, and a C client API exists.

NEOS [25] is a network-enabled problem-solving environment designed as a generic application service provider (ASP). Any application that can be changed to read its inputs from files, and write its output to a single file can be integrated into NEOS. The NEOS Server acts as an intermediary for all communication. The client data files go to the NEOS server, which sends the data to the solver resources, collects the results and then returns the results to the client. Clients can use email, web, sockets based tools and CORBA interfaces.

Other projects are related to various aspects of NetSolve. For example, task farming style computation is provided by the Apples Parameter Sweep Template (APST) project [26], the Condor Master Worker (MW) project [27], and the Nimrod-G project [28]. Request sequencing is handled by projects like Condor DAGman [6].

However, NetSolve provides a complete solution for easy access to remote resources and software. It differs from the other NES implementation by including a tight, simple integration with a variety of client PSEs (Matlab, Mathematica, Octave). Interface descriptions for a variety of standard mathematical libraries is distributed with NetSolve, and it is easy for additional services to be added. The ability to use server-proxies to leverage additional resource management and scheduling environments also adds to NetSolve's strengths.

## 7 Conclusion

Since the inception of the NetSolve project, the paramount design goal has been ease of use. This has motivated us to develop interfaces to popular interactive scientific computing environments such as Matlab, Mathematica, and Octave. The existence of these interfaces serves to reduce the barrier to entry for Grid computing since users do not need significant programming expertise to use them. NetSolve's agent based scheduling eliminates the need for the user to know the location of resources capable of servicing the request and the fault tolerance mechanism allows selecting alternate resources without intervention from the user. The NetSolve server also hides complexity from the user by making it easier to invoke parallel programs or jobs on machines controlled by a batch queue.

We have described how NetSolve is being used by several projects to solve problems ranging from medical imaging to optimizing the emissions from diesel engines. As researchers continue to investigate feasible ways to harness computational resources, we hope the NetSolve system will continue to provide a useful paradigm for Grid computing.

## 8 Acknowledgments

We would like to thank the following people for describing the status of their research using NetSolve: Jens Gregor, Lou Gross, Tomo Hiroyasu, Michael Langston, Zhiao Shi, and Michael Thomason.

## References

[1] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova. Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In M. Parashar, editor, *GRID 2002*, pages 274–278, 2002.

[2] Global Grid Forum Research Group on Programming Models. `http://www.gridforum.org/7_APM/APS.htm`.

[3] Hidemoto Nakada, Mitsuhisa Sato, and Satoshi Sekiguchi. Design and Implementations of Ninf: Towards a Global Computing Infrastructure. In *Future Generation Computing Systems*, *Metacomputing Issue*, volume 15, pages 649–658, 1999.

[4] 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.

[5] Load Sharing Facility. `http://www.platform.com/products/LSF/`.

[6] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steve Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5:237–246, 2002.

[7] Z. Chen, J. Dongarra, P. Luszczek, and K. Roche. Self Adapting Software for Numerical Linear Algebra and LAPACK For Clusters. In *Parallel Computing*, volume 29, pages 1723–1743, 2003.

[8] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4.1. Innovative Computing Laboratory. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.

[9] A. Bassi, M. Beck, T. Moore, J. Plank, M. Swany, R. Wolski, and G. Fagg. The Internet Backplane Protocol: A Study in Resource Sharing. In *Future Generation Computing Systems*, volume 19, pages 551–561.

[10] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. A GridRPC Model and API for End-User Applications. `http://forge.gridforum.org/projects/gridrpc-wg/document/End-User_API_23%_Sept_04/en/1`, September 2004. Global Grid Forum Recommendation Draft.

[11] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 1997.

[12] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631, May 1994.

[13] K. Moore. Recommendations for the Design and Implementation of NAT-Tolerant Applications. Internet-draft, February 2002. Work in Progress.

[14] D. M. Fleming, D. L. DeAngelis, L. J. Gross, R. E. Ulanowicz, W. F. Wolff, W. F. Loftus, and M. A. Huston. ATLSS: Across-Trophic-Level System Simulation for the Freshwater Wetlands of the Everglades and Big Cypress Swamp. National Biological Service Technical Report, 1994.

[15] M. Beck, Y. Ding, S. Atchley, and J. S. Plank. Algorithms for High Performance, Wide-area Distributed File Downloads. *Parallel Processing Letters*, 13(2):207–224, June 2003.

[16] Statistical Parametric Mapping. `http://www.fil.ion.ucl.ac.uk/spm/`.

[17] G. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996.

[18] 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, Third Edition*. SIAM, Philadelphia, PA, 1999.

[19] Direct Internet Message Encapsulation. `http://www.gotdotnet.com/team/xml_wsspecs/dime/dime.htm`.

[20] Scalable Intracampus Research Grid. `http://icl.cs.utk.edu/sinrg`.

[21] H. Hiroyasu, T. Hiroyasu, M. Miki, J. Kamiura, and S. Watanabe. Multi-Objective Optimization of Diesel Engine Emissions using Genetic Algorithms and Phenomenological Model. *JSAE (Society of Automotive Engineers of Japan) Paper No. 20025489*, November 2002.

[22] H. Hiroyasu and T. Kadota. Models for Combustion and Formation of Nitric Oxide and Soot in Direct Injection Diesel Engines. *SAE Paper 760129*, 1976.

[23] Y. Tanaka, H. Nakada, S. Sekiguchi, Suzumura Suzumura, and S. Matsuoka. Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing. *Journal of Grid Computing*, 1(1):41–51, 2003.

[24] E. Caron, F. Desprez, F. Lombard, J.-M. Nicod, L. Philippe, M. Quinson, and F. Suter. A Scalable Approach to Network Enabled Servers (Research Note). *Lecture Notes in Computer Science*, 2400, 2002.

[25] E. Dolan, R. Fourer, J. J. Moré, and Munson Munson. The NEOS Server for Optimization: Version 4 and Beyond. Technical Report ANL/MCS-P947-0202, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, February 2002.

[26] Henri Casanova, Graziano Obertelli, Berman Berman, and Rich Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proceedings of Supercomputing'2000 (CD-ROM)*, Dallas, TX, Nov 2000. IEEE and ACM SIGARCH.

[27] Jeff Linderoth, Sanjeev Kulkarni, Jean-Pierre Goux, and Michael Yoder. An Enabling Framework for Master-Worker Applications on the Computational Grid. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pages 43–50, Pittsburgh, PA, August 2000.

[28] David Abramson, Rajkumar Buyya, and Jonathan Giddy. A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. *Future Generation Computer Systems*, 18(8):1061–1074, October 2002.