Recent Developments in GridSolve

Asim YarKhan Keith Seymour Kiran Sagi Zhiao Shi Jack Dongarra
Department of Computer Science
University of Tennessee, Knoxville
Knoxville, TN 37996

October 3, 2005

Abstract

The purpose of GridSolve 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. In addition to supporting a diverse set of hardware, such as desktop computers, clusters, and massively parallel computers, Grid middleware may need to interact with the software managing those systems, such as Condor, LFC (LAPACK for Clusters), and batch queues. Furthermore, user requests may be characterized in different ways (parameter sweep, task graph, etc.), each with different requirements. This diversity has led us to implement scheduling in different layers of GridSolve with the understanding that a strategy for scheduling and resource allocation is an essential part of realizing the vision of transparent Grid computing. In this paper we will discuss some of these scheduling mechanisms and some of the possible interactions with external systems such as LFC and Condor.

1 Introduction

The emergence of Grid computing as the prototype of a next generation cyber infrastructure for science has generated high expectations for its potential as an accelerator of discovery. However, it has also raised questions about whether and how the broad population of research professionals, who must be the foundation of such productivity, can be motivated to adopt this new and more complex infrastructure. The rise of the new era of scientific modeling and simulation has, after all, been precipitous, and many science and engineering professionals have only recently become comfortable with the relatively simple world of the uniprocessor workstations and desktop scientific computing tools. In that world, software packages such as Matlab and Mathematica represent general-purpose scientific computing environments that enable users - totaling more than a million worldwide - to solve a wide variety of problems through flexible user interfaces that can model in a natural way the mathematical aspects of many different problem domains. Moreover, the ongoing, exponential increase in the computing resources supplied by the typical workstation makes these scientific computing environments more and more powerful, and thereby tends to reduce the need for the kind of resource sharing that represents a major strength of Grid computing. Certainly there are various forces now urging collaboration across disciplines and distances, and the burgeoning Grid community, which aims to facilitate such collaboration, has made significant progress in mitigating the well-known complex-

ities of building, operating, and using distributed computing environments. But it is unrealistic to expect the transition of research professionals to the Grid to be anything but halting and slow if it means abandoning the scientific computing environments that they rightfully view as a major source of their productivity.

The GridSolve project addresses this difficult problem directly: The purpose of GridSolve 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. 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 GridSolve is designed to realize.

To that end, GridSolve employs NetSolve [1] as one of its primary enabling technologies. NetSolve is a client-agent-server system which provides remote access to hardware and software resources through a variety of client interfaces.

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

- The *Client*, which needs to execute some remote procedure call. In addition to C and Fortran programs, the NetSolve client may be an interactive problem solving environment such as Matlab or Mathematica.

- 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 function 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, GridSolve 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 act as 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 [2]. We briefly discuss these two approaches here.

Normally the GridSolve server executes the actual service request itself, but in some cases it can act as a proxy to other services such as Condor. The primary benefit is that the client-to-server communication protocol is identical so

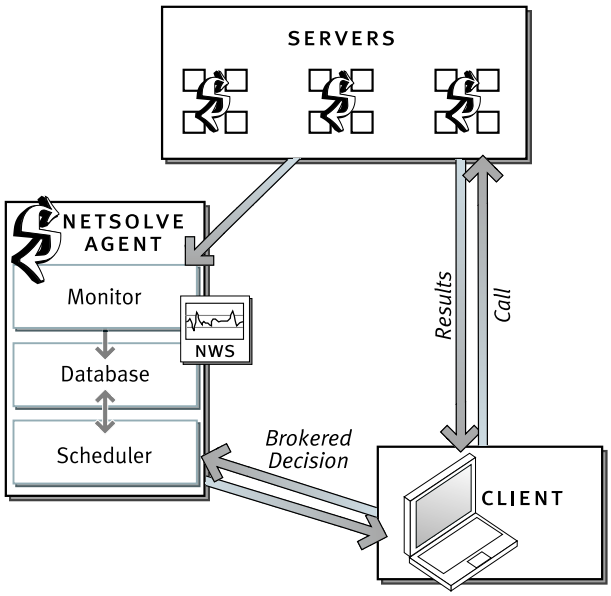

Figure 1: Overview of NetSolve

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 GridSolve server. We will discuss this in more detail in Section 2.4.

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 [3]. The initial work on GridRPC reported in [2] shows that client access to existing Grid computing systems such as NetSolve and Ninf [4] can be unified via a common API, a task that has proven 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.

The combination of these technologies will allow GridSolve to provide seamless client access to a diverse set of Grid services. Since GridSolve encompasses NetSolve and to avoid confusion, we will hereafter only use the term GridSolve.

2 Scheduling in GridSolve

In this section we discuss several approaches to scheduling that have been implemented in the GridSolve 2.0 distribution.

2.1 Agent Based Scheduling

In agent based scheduling, the 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

```
for all servers S_i that can provide the desired service
    T_1(S_i) = estimated amount of time for computation on S_i
    T_2(S_i) = estimated time for communicating input and output data
    T(S_i) = T_1(S_i) + T_2(S_i) estimated total time using S_i
select the server S_m which has the minimum time, where T(S_m) = min T(S_i) ∀i
```

Figure 2: Minimum Completion Time algorithm

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 time to completion, and then returns the list of servers to the client. The client then sends the service request to the fastest server. If that fails for some reason, the client can submit the service request to the next fastest service, thus providing a basic level of fault tolerance. This scheduling heuristic, summarized in Figure 2, is known as *Minimum Completion Time* and it works well in many practical cases. Each service request should 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.

To evaluate the effectiveness of this form of load balancing, we submit 16 `DGEMM` (matrix multiply) requests to a GridSolve Grid with a varying number of active servers. The servers run on a cluster of dual processor 933MHz Pentium 3 machines and the agent and client run elsewhere on the network. As Figure 3 illustrates, when more servers are present the GridSolve agent can balance the load among the available servers resulting in lower overall execution time. The scalability is not perfect since adding servers only reduces the total computational cost, not the communication cost.

However, the Minimum Completion Time heuristic does not try to maximize the throughput when servers are allowed to run multiple services and there are many more requested services than available servers. Since an estimate of the execution time for currently executing service is available, this knowledge could be used to schedule new service requests more intelligently. Some explorations of alternative scheduling heuristics using historical execution trace information in are described in [5].

2.2 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 GridSolve agent assigns the entire sequence to a single server. The server is selected based

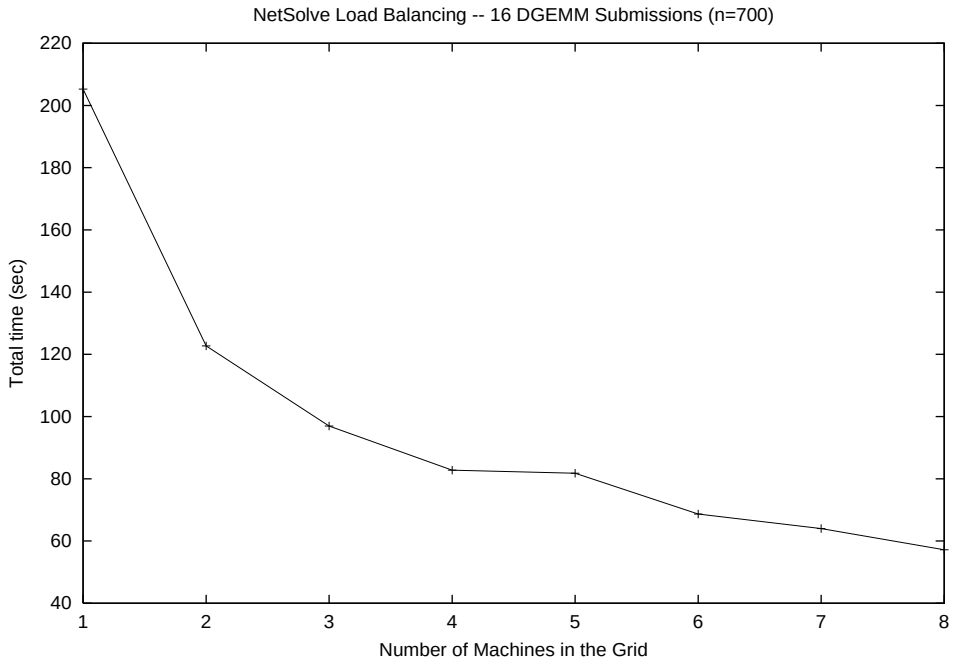

Figure 3: GridSolve Load Balancing with Agent-based Scheduling

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 GridSolve server, GridSolve 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 GridSolve 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 GridSolve requests, each dependent on the results of the previous request: `DGEMM` $\Rightarrow$ `DLACPY` $\Rightarrow$ `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. Using the same machines as in the previous experiment, submitting this sequence as three individual requests took 4.43 seconds on average, while submitting it as one sequence only took 3.07 seconds. 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.

2.3 Task Farming in GridSolve

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 naive algorithm could be implemented by using the standard GridSolve interface and letting the GridSolve 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 GridSolve goal of ease-of-use.

In order to provide an efficient and easy to use interface to task farming, GridSolve 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 4 shows an example of the task farming interface. The task farming API only adds 4 calls to GridSolve, namely 3 calls for constructing arrays of different data types, and 1 call for the actual farming. More details about the API can be found in the Users Guide to GridSolve [1].

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 GridSolve.

2.4 Server Based Scheduling

Part of the GridSolve philosophy is to provide easy and transparent interfaces to access and reuse existing software solutions. In line with this, GridSolve provides mechanisms to provide interfaces to alternative scheduling and execution systems.

In the server based approach to scheduling, GridSolve creates server-proxies to delegate the scheduling to specialized scheduling and execution services such as batch systems, Condor or LFC (LAPACK for Clusters). The GridSolve agent sees the server-proxy as a single server entity, even though the server-proxy can represent a large number of actual resources, and so the proxy handles the scheduling for these resources, rather than the GridSolve agent.

The GridSolve agent can decide to send the service request to a server-proxy based on several factors (e.g., the proxy can register itself with the agent as a virtual server with a large amount of processing power). The server-proxy will delegate the request to the specialized service (e.g. Condor), which schedules and executes the request. The server-proxy then returns the results back to the client.

2.4.1 LAPACK for Clusters

The integration of LAPACK For Clusters (LFC) [6] into the GridSolve system gives the GridSolve system the ability to access clusters more efficiently. It allows the GridSolve user to access LFC software, which attempts to optimally use the resources of the cluster, via C and Matlab programming interfaces.

The LFC software developed at the University of Tennessee, Knoxville, exposes a serial, single processor user interface, but delivers computing power achievable by running the same problem in parallel on a set of resources of a cluster. It allows the user to call LFC routines from a serial environment, addresses computational time and space complexity issues and maps the problem into a parallel environment if it is possible to execute the problem in less time

Using standard non-blocking GridSolve 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"));
```

Figure 4: 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).

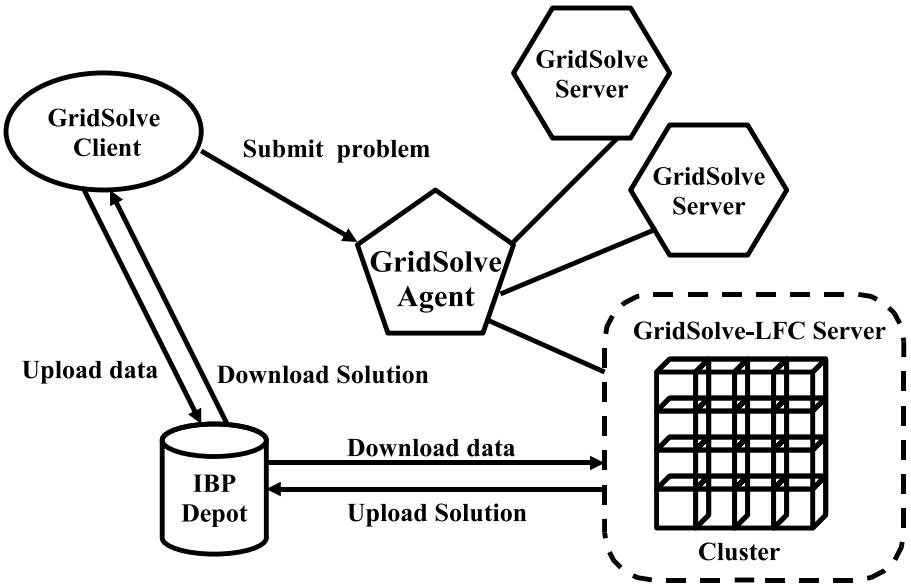

Figure 5: GridSolve with LFC (LAPACK For Clusters)

in parallel.

Figure 5 illustrates how the GridSolve system utilizes cluster resources through LFC. One of the machines in the cluster is chosen to be the specialized GridSolve-LFC server-proxy. When this specialized server is started, it runs a benchmark on the cluster and reports the computational power and workload information to the GridSolve agent. At present we use the sum of KFlops of all the machines of the cluster to represent the computational power of the cluster, but this can be refined to make the GridSolve scheduler more intelligent. The LFC server-proxy also updates the GridSolve agent with changes in workload and communication costs on the cluster at regular intervals.

The GridSolve user, who wants to solve a problem, prepares the input data and uploads the data into a remote network storage using the Internet Backplane Protocol (IBP) [7] API. IBP is middleware for managing and using remote storage. Handles for accessing the uploaded data are returned from IBP to the user. The GridSolve user then submits the problem to the GridSolve system. The GridSolve agent uses the problem size and complexity of the problem to determine the computational cost of the problem. It chooses the best available server for solving the problem based on the available servers. and their computational power, workload and communication costs. If the problem request is sent to the GridSolve-LFC specialized server, it makes a LFC routine call to schedule the problem on the best subset of resources of the cluster. The details of parallelizing the user's problem, selecting the parallel algorithm to be used, resource discovery, selection, allocation, downloading the data from the IBP depot using the IBP handles, mapping the data onto the working cluster of processors are handled by the LFC software. LFC executes the desired service and writes the solution into the IBP network storage depot. The user downloads the solution from the network storage using IBP handles.

GridSolve users can use C and Matlab programming interfaces to submit problems to the GridSolve-LFC specialized server. In future, these interfaces will be extended to include Fortran, Mathematica and Octave.

2.4.2 Condor and Condor-G

Condor [8] provides a high throughput environment for running compute-intensive jobs. Condor includes a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Condor-G [9] is an implementation of Condor which is interoperable with resources in a Globus [10] environment.

A server-proxy has been developed to enable a GridSolve client to send a service request to a resource pool that is managed by Condor or Condor-G. The server-proxy must be part of the Condor pool itself. The server-proxy takes the service-request, creates files containing the input parameters, and creates a Condor command file which will execute the desired service. The command file is submitted to the Condor job manager, which handles the details of scheduling and executing the service-request on the appropriate set of resources in the Condor pool. In the case of Condor-G, a Globus job manager must be specified by the proxy.

Figure 6 shows the interaction between GridSolve and Condor-G. In the figure, a GridSolve server-proxy, which is part of the Condor pool, makes all the services on that server available as remote Grid services. The GridSolve client user does not need to have Globus certificates to access those services. However, the service provider needs to enable the server-proxy to execute service requests on the Globus resources by providing the server-proxy with valid Globus credentials (e.g. the service provide can issue a grid-proxy-init for the proxy-server). The server-proxy can grant or deny client access by using standard GridSolve access control mechanisms that use Kerberos. When a job is submitted to Condor-G server, a temporary submit description file similar to the following will be prepared by the server as shown in Figure 7.

The `globusscheduler` command is depen-

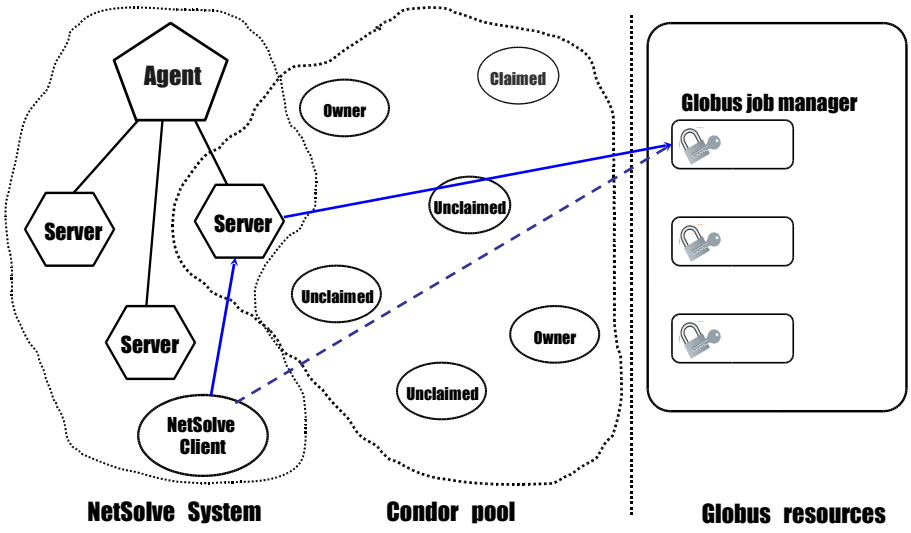

Figure 6: GridSolve with Condor-G

```
executable = homes/user/GridSolve/bin/i686_pc_linux_gnu/service-lapack_subset
globusscheduler = machine03.cs.utk.edu/jobmanager
universe = globus
output = condor.out
error = condor.error
arguments = 0 /homes/user/GridSolve/bin/i686_pc_linux_gnu
queue
```

Figure 7: GridSolve's Condor-G submit file

dent on the scheduling software available on remote resource. This required command should be changed based on the Grid resource intended for execution of the job. The Condor-G job is then submitted to the Globus universe. The GridSolve server issues `condor_submit` to submit the job for execution on Globus resources. After the job is finished on remote machines, the server-proxy will collect the results and send them back to the client.

Similar to GridSolve's request sequencing, Condor [8] has a meta-scheduler called DAGMan (Directed Acyclic Graph Manager) [11], which can be used to facilitate data flow between requests . The user creates a DAGMan input file which describes the dependencies and other information about the requests and submits it to Condor for scheduling. While GridSolve has the capability to submit single jobs to Condor, it would be useful to extend this capability to allow submitting sequenced requests. Since GridSolve already builds a DAG internally (in its own format) it should be feasible for GridSolve to create a DAGMan input file and submit the entire sequence. This would provide the various GridSolve clients an easy way to submit sequenced requests to resources managed by Condor.

3 Extensions to GridSolve

Over time, many enhancements have been made to GridSolve to extend its functionality or to address various limitations including task farm-

ing, request sequencing, and security. However, some desirable enhancements cannot be easily implemented within the current GridSolve framework. Thus, our ongoing work on GridSolve involves redesigning the framework from the ground up to address some of these new requirements.

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

The overall goal is to address three general problems: ease of use, interoperability, and scalability. Improving ease of use primarily refers to improving the process of integrating user code and libraries into a GridSolve server. Interoperability encompasses several facets, including better handling of different network topologies, better support for parallel libraries and parallel architectures, and better interaction with other Grid computing systems such as Globus [10] and Ninf [4]. 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.

3.1 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 *proxy*, which is a component that allows servers to exist behind a NAT. Since a server cannot 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.

3.2 Scheduling Enhancements

The next generation of 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 GridSolve system, the only user-provided filter that affects the selection of resources is the problem name. Given the problem name, the GridSolve 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 the next generation of 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.

3.3 IDL Improvements

One of the original design goals of GridSolve 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 GridSolve requires 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 De-

scription File format does not support it in a general way. Additionally, some parallel software has required using a customized GridSolve 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.

4 Related Work

Several Network Enabled Servers (NES) provide mechanisms for transparent access to remote resources and software. Ninf-G [14] is a reference implementation of the GridRPC API [2] built on top of the Globus Toolkit. Ninf-G provides an interface definition language that allows services to be easily added, and client binding are available in C and Java. Security, scheduling and resource management are left up to Globus.

The DIET (Distributed Interactive Engineering Toolbox) project [15] 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 [16] 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 GridSolve. For example, task farming style computation is provided by the Apples Parameter Sweep Template (APST) project [17], the Condor Master Worker (MW) project [18], and the Nimrod-G project [19]. Request sequencing is handled by projects like Condor DAGman [9].

However, GridSolve 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 GridSolve, and it is easy for additional services to be added. The ability to use server-proxies to make it easy to leverage additional resource management and scheduling environments also adds to GridSolve's strengths.

5 Conclusion

One of the most important aspects of the middleware we have presented here is the scheduling that maps user requests to the most appropriate resource. Scheduling requests in a Grid environment is not trivial and we find that different forms of scheduling are useful under different circumstances. The traditional agent based scheduling provides a coarse load balancing among all the available resources. Server based scheduling is useful for scheduling tasks on specialized or aggregate resources (such as clusters and Condor pools) while providing a consistent interface to the various clients. Request sequencing schedules an entire task graph on one server, which is appropriate when there is a lot of intermediate data that we want to avoid transferring between the client and server. Task farming is useful for parameter sweep applications where a large number of independent, simultaneous requests are to be scheduled. It is worth noting that these forms of scheduling are not exclusive and we typically employ a combination of these approaches within one Grid.

Ongoing work on scheduling in GridSolve involves client-based scheduling, which includes filtering based on client-specified resource selection criteria as well as providing information to

the client to allow full fledged scheduling. We are also planning to extend task sequencing to allow tasks in the DAG to be scheduled on different servers if there is any parallelism that can be exploited. This would involve not only scheduling of computational resources, but could involve scheduling of data storage and transfer as well.

References

[1] 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.

[2] K. Seymour, N. Hakada, 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.

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

[4] 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.

[5] Yves Caniou and Emmanuel Jeannot. Experimental study of multi-criteria scheduling heuristics for GridRPC systems. In *ACM-IFIP Euro-Par 2004*, Pisa, Italy, Sept 2004.

[6] 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.

[7] 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.

[8] Douglas Thain and Miron Livny. Building reliable clients and servers. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[9] 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.

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

[11] Condor DAGMan. `http://www.cs.wisc.edu/condor/dagman`.

[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] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.