Developing an Architecture to Support the Implementation and Development of Scientific Computing Applications

Dorian C. Arnold and Jack J. Dongarra
Computer Science Department
University of Tennessee
Knoxville, TN 37996
[darnold, dongarra]@cs.utk.edu

Abstract

As scientific computing applications continue to become more complex, it has become apparent, now more than ever, that there is a need for robust software architectures to facilitate the conceptualization, design, implementation, deployment and maintenance of these applications. This paper attempts to shed light on how the unique characteristics of scientific computing applications, as well as computational scientists, make impositions upon the framework used to support research efforts. We use our experience with NetSolve, a toolkit designed just for such interactions, as a means to present the approach of one infrastructure intended to support scientific computing and show how it implements the unique model of using a single system to aggregate, manage and access distributed hardware *and* software resources.

1 Introduction

Scientists and engineers have become almost completely reliant on the computer as a tool for advanced modeling and simulation of their experiments and analyses. Simulating anything, from the electromagnetic field of a difribulator in a virtual human to warfare scenarios with tens of thousands of components interacting together, is now possible thanks to advancements in computer technology. While the rapid rate of microprocessor performance growth has been constant, decades ago scientists realized that an intuitive way to increase the computational capacity of any single processor is by connecting multiple processors together. Indeed, the scientific computing application (SCA) that runs solely on a single processor has become a rare breed. Supercomputers, with massively parallel processors (MPP) or shared memory multi-processors (SMMP), are very common and today's fastest computers operate at speeds of over 2 Tflop/s. These speeds have been achieved by improved chip technology and computing architectures that support parallel computing. While the innovations of the computer engineers are increasing the speeds of microprocessors and developing new paradigms to use them together, computer scientists are exploring ways to increase computational capacities and capabilities via software infrastructures. However, the computational scientist is now faced with highly complex machines and the fact that both the accuracy and performance of his code depend upon the level of his knowledge of the architecture(s) at hand. Furthermore, since every SCA demands interdisciplinary expertise (mathematics, computer science, and the domain science at the very least,) there must be simplistic ways for the computational scientist to leverage the effort of algorithm developers, software system designers and hardware engineers and merge them with his own specialties.

The emergence of technologies like PVM [1] and standards like MPI [2] show strong efforts to develop a common parallel programming environment. Even more recently, there has been the emergence of the Grid [3] and Grid computing concepts that envision software technologies exploiting today's high network connectivity to create a single, global virtual machine. However, little attention has been focused on the computational scientists and what is required for them to accomplish useful tasks without the colossal nightmare of becoming a computer scientist and a mathematician (and maybe even a magician) overnight. The goal of this article is to analyze the unique needs of SCAs and the people that develop them in an attempt to establish the issues that supporting software infrastructures should resolve. In the following, we also discuss the NetSolve distributed computing environment which attempts to address these issues in a practical and efficient way. Section 2 defines what we have found to be the fundamental characteristics of scientific computing that should shape the supporting scientific computing application infrastructure (SCAI). After a general overview of the NetSolve system in Sect. 3, Sect. 4 evaluates some of the features of NetSolve based on the gathered requirement of scientists and their applications. What we hope to present is not only an introduction of NetSolve, but a means by which to evaluate any infrastructure that claims to support scientific computing based on the needs of that community.

2 Characteristics of Scientific Computing

Here, we talk about some of the common characteristics of computational scientist and their SCAs that should impact the SCAIs the applications use.

Scientific applications typically have four phases as depicted in Fig. 1. Regardless of whether the application is Graphical User Interface (GUI) based or text based, the first phase entails the gathering and preparation of input data. This may mean getting user input, allocating memory requirements, constructing specialized data structures and the like. After the input is made ready, it is passed to the computationally intensive phase of the process when complex algorithms are run on the prepared data. After the data processing, often there is an analysis phase which may be used to determine if further processing of the data is necessary, among other things. The data processing and analyses phases may undergo several iterations until finally some output results are made available. This may be in the form of textual output or graphical images that visualize the simulation and/or its results.

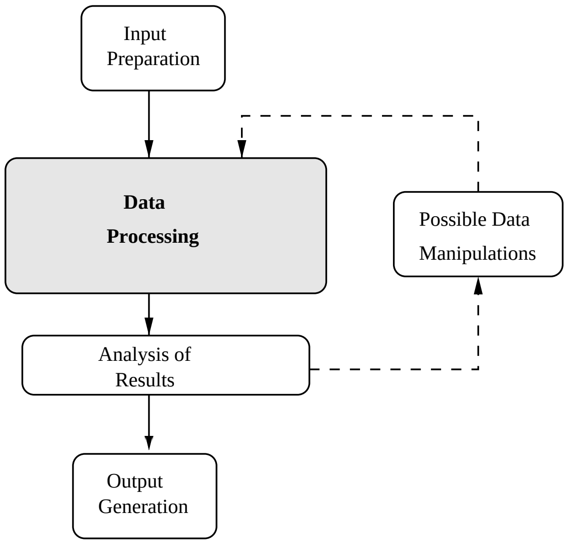

Figure 1: The four typical phases of a scientific computing application

To this point, we have made no mention of possible parallelizations that may take place in the program. Typically, the only difference between parallel applications and serial ones is that in a parallel application, the data processing phase is distributed amongst multiple processors. From this point on, we assume the more common scenario of a parallel application. Parallel applications can be generally categorized into two classes of applications; this categorization depends on whether or not there is communications amongst the computational nodes during the computation. Figure 2 depicts what we will refer to as cooperative parallelism where there is interprocess communication amongst the computational nodes. Figure 3 shows the other kind of parallelism where there are concurrently running modules on multiple processors, but no communications amongst the nodes; we term this independently parallel execution.

The common (and relevant) characteristics of these SCAs and their developers are described below. These have been collected from a variety of sources, including [4] in an attempt to provide a complete picture of the demands of scientific computing. The format of the items that follow is a detailed description of the characteristic followed by a single (*emphasized*) sentence that summarizes the impact that characteristic has on SCAIs:

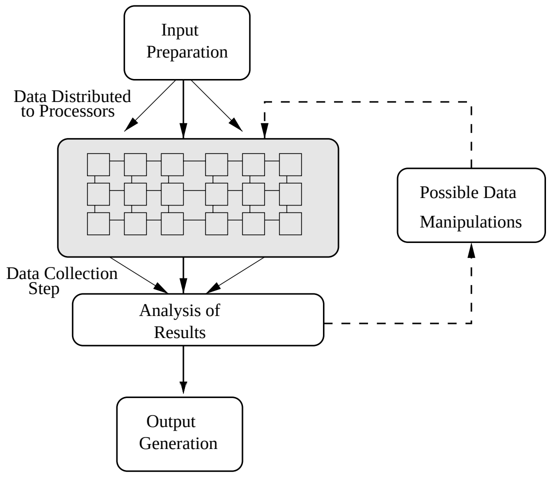

Figure 2: The parallel scientific computing application with cooperative parallelism

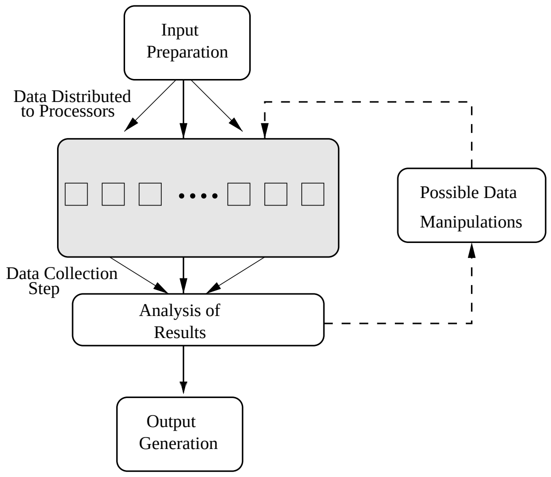

Figure 3: The parallel scientific computing application with independent parallelism

•Knowledge Base of the Scientist
Whether it be a computational chemistry or nuclear engineering application, the computational scientist will have a deep knowledge and understanding of the concepts of their scientific domain. Often, they also possess a thorough understanding of relevant mathematical concepts. However, their expertise in programming and other aspects of software engineering is typically limited. In larger collaborations, the application development is many times carried out by a concert of domain scientists, computer scientists and, perhaps, mathematicians. Not every organization and research effort can afford such luxuries, nor do they wish to.
The SCAI must provide an easy and intuitive programming model that allows computational scientist to integrate complex supporting software and highly-optimized mathematical modules without advanced levels of expertise.

•Composition of Existing Software Components
To help alleviate the previous problem, the practice of providing highly optimized numerical software libraries is very common. Numerical solvers (linear/non-linear systems solvers, partial differential equation (PDE) solvers, eigensolvers, etc.) are the fundamental subprograms found in practically all SCAs. Packages like LAPACK [5], PETSc [6] and ARPACK [7] are only a few examples of such packages that implement sophisticated techniques. As is the case with these examples, a wide array of this software is freely available, and even open-sourced. For obvious reasons, the domain scientist will want to leverage the years, if not decades, of research and development incorporated in the older packages or the novel concepts and capabilities of the newer ones.

The SCAI must provide ways to easily incorporate largely varied types of computational modules.

•**Granularity of Computational Modules**
The granularity of the computational modules can be defined as the relation between the floating point performance of the module and the average communication bandwidth of a single processor (Flop/Byte) [8]. A higher granularity means that the computational modules take on large, atomic chunks of work at a time, while a lower granularity implies that not much processing takes place before data is transported. The granularity of the computational modules can depend upon the class to which that module belongs. Lower level classifications like a mathematical linear system solver (or even the matrix multiply routine upon which the solver depends) leads to low granularities. Higher level classifications like a data transformation module that incorporates linear systems solvers, or an image processing module that uses a series of data transformations yield higher granularity. Depending on the specific application, it may be warranted to use a large granularity, a small one, or both.
Varying granularities of computational modules must be supported; neither performance nor ease-of-integration should depend upon granularity.

•**Too Many Choices**
As can be seen, there are many software packages available from a variety of sources. While this allows for much flexibility when deciding which packages to use, it also implies climbing a steep learning curve to determine the peculiarities of each package to make a fair evaluation of which suit the purposes. The veritable alphabet soup of packages available can be daunting, yet discovering which package to use is the least of the troubles. As is the case of iterative methods, an algorithm is only as good as the manner in which you use it. Veterans of iterative methods agree that finding the right combination of solver, pre-conditioner, scaling and re-ordering is an art form developed only from experimentation within the application area [9].
The SCAI must hide the complexities and intricacies of the underlying computational modules while providing convenient ways for the scientist to discover what services he wants.

•**Problem Capacity** Often, the memory requirements or the processing demands of the SCA exceed the capacity of workstations and requires multiple processors to accurately and efficiently solve the problem specifications. In other cases, it may be that the only appropriate algorithms discovered by the computational scientist involve code that only run on a particular server. Or, combining these two scenarios, the only feasible solvers are specialized parallel algorithms on a remote distributed memory MPP system [9].
The SCAI must be able to execute modules on remote servers in a reliable and efficient way.

•**Interaction Levels**

Although there definitely is a place for GUIs in rapid prototyping and quick experimentation with SCAs, generally, these high-performance computing applications can take hours, days and weeks to complete a single task. The user may not want to keep his GUI open for that duration and definitely does not want to be required to interact for that period. And while GUIs at times provide a more convenient programming model, the day has not yet arrived when a graphical programming environment can implement a doubly nested `for` loop executing a module hundreds of times more conveniently than a scripting language.
Scripting capabilities must be available to allow users to chain complex combinations of modules with as little sacrifice of convenience and ease-of-use as possible.

•Code Maintenance and Enhancement
The fact that the FORTRAN programming language is still popular after nearly 50 years implies two fundamental things: i) the laws of inertia apply to computer science as well (i.e. scientists and programmers are reluctant to diverge from a platform that can work for their purposes after tremendous time and efforts have been invested) and ii) it is extremely important for codes to be backward compatible and have a lifetime as long as possible. As a result, mixed-language programming is very common in SCAs – hybrid applications now integrate C, FORTRAN and object-oriented languages like Java and C++ to exploit the different advantages of each platform whether it be performance, maintenance, security or programming methodologies (or, simply, code availability).
The SCAI must have multiple-language support and provide mechanisms which allow users to easily replace components with newer, more optimized, or simply corrected, versions without significant code modification.

•Criticality of Performance
Last, but by no means least, SCAs demand optimal performance. It is the very nature of these high-performance applications to consume and dominate any and all resources they can access and still require days and weeks of computational time to complete. Data sets can easily extend into the hundreds of megabytes and gigabyte range for a single application, and this too needs to be considered, especially when data transfer to remote servers are involved.
The design of the SCAI must consider the performance impact of its component interactions with respect to the tasks/services they are providing, as well as the overhead of the SCAIs components place on a host (outside the context of solving a particular problem), especially when the machine is not dedicated solely to interaction via the SCAI.

3 The NetSolve Computational Environment

The NetSolve project is being developed at the University of Tennessee's Innovative Computing Laboratory of the Computer Science Department. Its original motivation was to alleviate the difficulties that domain scientists encounter when trying to locate/install/use numerical software, especially on multiple platforms. Today, the name NetSolve has become a misnomer, as the system has evolved into much more than a way to access numerical solver routines. NetSolve provides an environment that monitors and manages computational resources, both hardware and software, and allocates the services they provide to NetSolve-enabled client programs. It incorporates load balancing and scheduling strategies to distribute tasks evenly amongst servers. Built upon standard Internet protocols, like TCP/IP sockets, it is available for all popular variants of the UNIX operating system, and parts of the system are available for the Microsoft Windows '95, '98, '00 and NT platforms.

Figure 4 shows the infrastructure of the NetSolve system and its relation to the applications that use it. NetSolve and systems like it are often referred to as Grid middleware; this figure helps to make the reason for this terminology clearer. The shaded parts of the figure represent the NetSolve system. It can be seen that NetSolve acts as glue layer that brings the application or user together with the hardware and/or software it needs to complete useful tasks.

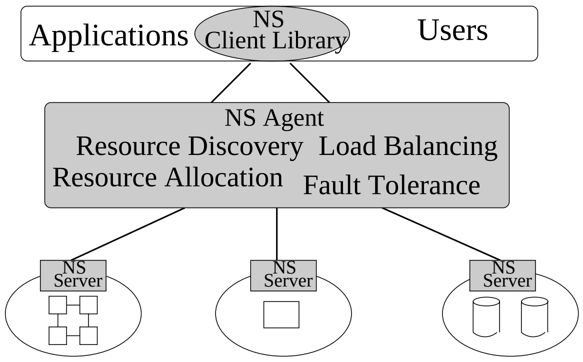

Figure 4: Architectural Overview of the NetSolve System

At the top tier, the NetSolve client library is linked in with the user's application. The application then makes calls via NetSolve's application programming interface (API) for specific services. Through the API, NetSolve client-users gain access to aggregate resources without the users needing to know anything about computer networking or distributed computing. In fact, the user does not even have to know remote resources are involved. NetSolve currently supports the C, FORTRAN, Matlab, and Mathematica programming interfaces as languages of implementation for client programs.

The NetSolve agent represents the gateway to the NetSolve system. It maintains a database of NetSolve servers along with their capabilities (hardware performance and allocated software) and dynamic usage statistics. It uses this information to allocate server resources for client requests. The agent, in its resource allocation mechanism, attempts to find the server that will service the request the quickest, balance the load amongst its servers and keep track of failed servers. Requests are directed away from failed servers. The agent also adds fault-tolerant features that attempt to use every appropriate server until it finds one that successfully services the request.

The NetSolve server is the computational backbone of the system. It is a daemon process that awaits client requests. The server can run on single workstations, clusters of workstations, symmetric multi-processors or machines with massively parallel processors. A key component of the NetSolve server is a source code generator which parses a NetSolve problem description file (PDF). This PDF contains information that allows the NetSolve system to create new modules and incorporate new functionalities. In essence, the PDF defines a wrapper that NetSolve uses to call the function being incorporated.

3.1 The Status Of NetSolve

Version 1.3 was released in May of 2000. Features implemented in this release include a Java GUI to aid in the creation of PDFs, a Microsoft Excel interface, more object data types, more server modules included with the distribution, and enhanced load balancing among other things. NetSolve-1.3, including APIs for the Win32 platform, can be downloaded from the project web site at **www.cs.utk.edu/netsolve**. NetSolve has been recognized as a significant effort in research and development, and was named in R&D Magazine's top 100 list for 1999. The reader is directed to [10] for further details of the system not discussed in this article.

4 NetSolve and Scientific Computing

The NetSolve system, as mentioned in the previous section, was designed specifically with the computational scientist in mind. The goal was to make his job of advanced research in his domain easier by facilitating implementation and deployment. This section compares the system with the requirements of the scientific computing community as discussed in Sect. 2 by placing NetSolve's features alongside the corollaries of that discussion. While much effort has gone into the development of highly optimized software packages for the scientists and engineers to use, and many groups are researching ways to make more efficient use of aggregated hardware resources via software infrastructure, the NetSolve project is one of the only systems that integrates both these concepts. We believe that this is the key to a complete SCAI.

• **Corollary 1:** The SCAI should provide an easy programming model and API for novice non-computer scientists.

The NetSolve system allows for client users to embed functions from practically any software library into their applications without having to install, learn or maintain that package. With API's available to a wide variety of programming environments like C, FORTRAN, Matlab and Mathematica, the scientist can flexibly choose his language of implementation. Fig. 5 shows an example Matlab code, before and after the NetSolve API has been integrated. The code on the left hand-side is making a call to a Matlab native function to multiply two matrices `A` and `B` and store the result in `C`. The call to NetSolve, on the right hand-side of the figure, achieves the same end via the NetSolve framework.

```
...                          ...
A = load(input_matrix1);     A = load(input_matrix1);
B = load(input_matrix2);     B = load(input_matrix2);
C = matmul(A, B);            C = netsolve('matmul', A, B);
...                          ...
```

Figure 5: Sample Matlab code: Left side before NetSolve, right side after NetSolve integration

This example shows how NetSolve can be used to provide access to complicated software modules without expert interactions of the user. Apart from this module encapsulation, it allows one to create uniform interfaces of different packages of similar algorithms. For instance, the intricacies of the iterative methods of sparse solvers like PETSc [6], AZTEC [11] and others can hidden by a single common interface that takes an additional parameter defining the software package to use.

• **Corollary 2:** The SCAI must provide ways to easily incorporate largely varied types of computational modules.

As described in Sect. 3, the NetSolve system provides a code generator that parses a NetSolve PDF in order to extend the servers' functional capabilities. Figure 6 shows a segment of a PDF that was used to integrate a module from a sub-surface fluid simulator. The **PROBLEM** parameter of this file defines the name we want client applications to use when referring to this module. The **INCLUDE** and **LIB** directives are used in the compilation of the module. Among other things, this PDF eventually describes the code that determines how to call the simulation code with the inputs given from a client program. After this configuration and a compilation, the NetSolve server is ready to be attached to a NetSolve agent/system and service requests.

The PDF facility provides a way for NetSolve to seamlessly integrate any type of computational modules. Furthermore, we have created a Java GUI that makes this process even easier. One issue, however, is that the NetSolve system supports NetSolve objects (like the MATRIX, SPARSEMATRIX, VECTOR, FILE, etc.) and it is up to the author of the PDF to convert these objects into those supported by the code being integrated, as necessary.

• **Corollary 3:** The system should adapt to varying granularities of computa-

```
@PROBLEM ipars
@INCLUDE "ipars.h"
@LIB /home/user/lib/libipars.a
@DESCRIPTION
Parallel Sub-Surface Flow Simulator
@INPUT 2
@OBJECT STRING CHAR model
IPARS physical model to use
@OBJECT FILE CHAR infile
Input data file
....
```

Figure 6: Portion of a PDF used to integrate functional modules into the NetSolve system.

tional modules.

While the PDF facility does not concern itself with the computational module being integrated is finely or coarsely grained, an issue in any distributed environment is how the amounts and sizes of data transport affects performance. The first way NetSolve attempts to deal with this issue is by analyzing network bandwidths and latencies to choose the most conveniently located service resources to solve client requests. The second way we have optimized data communications is by creating an interface and infrastructure that allows a user to group or sequence a collection of NetSolve requests [12]. The system then analyzes the input and output parameters amongst all requests and caches common data near the relevant servers. Figure 7 illustrates the typical transactions that take place during a series of NetSolve requests by a single client. The important points to note are that parameter `A` is shared as an input for the first and second requests. Also, output parameters `C` and `D` serve as inputs for subsequent requests. Figure 8 shows the reduction in data flow that occurs when the sequencing mode is employed.

• **Corollary 4:** Complexities and intricacies of the underlying computational modules must be hidden, yet the modules must be conveniently accessible.

Through NetSolve, users are given access to complex algorithms that solve a variety of types of problems, one instance being linear systems solvers. All solvers, however, are not built alike; depending on the characteristics of the system being solved some perform poorly and others not at all. NetSolve has incorporated a large number of both direct and iterative solver algorithms for sparse/dense, symmetric/non-symmetric systems. To allow non-expert users to properly and efficiently use these algorithms without climbing the steep learning curve that would otherwise be involved, we have created an interface that allows

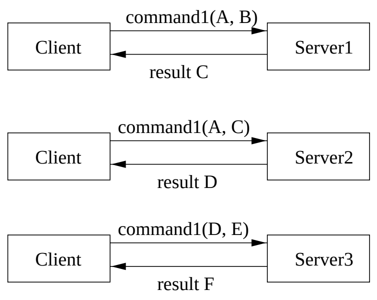

Figure 7: Client-server interactions during a typical request scenario.

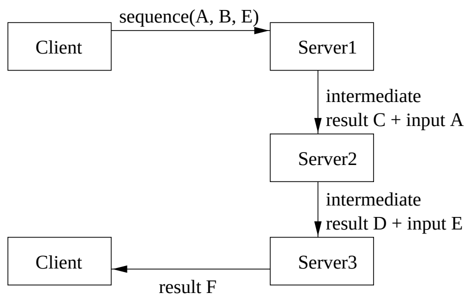

Figure 8: Client-server interactions during a "request sequence".

them to generically call a "LinearSolve" routine which transparently analyzes the input matrix and determines which algorithm to use based on input characteristics. [13] further discusses this interface and the heuristics and decisions that are involved in the algorithm selection process. We envision using similar heuristics for classes of problems other than linear system solvers in the near future.

• **Corollary 5:** The SCAI should reliably execute modules on remote servers.

By its very definition, NetSolve is a distributed computing environment that allows for remote problem execution. Its failure detection and fault-tolerant mechanisms allow the system to detect servers that have failed to solve particular problems or server hosts that are non-responsive and direct new requests away from these resources. During a computation, the system attempts to use every appropriate and capable server host (from best to worse (see Corollary 8)) until a problem has been solved or the list of servers has been exhausted. Other investigations are leading to the development of heuristics to checkpoint NetSolve services so that a mid-service failure does not result in a lost of all previous computation. These checkpoints will be used to migrate the state of the interrupted service to other NetSolve servers where computation will resume.

• **Corollary 6:** Scripting capabilities must be available to allow users to chain complex combinations of modules.

All elements of the NetSolve system are accessible via the API of the client libraries. Using the functions of this API, users can embed calls to NetSolve in compiled languages, like C or FORTRAN, or interpreted languages, like Matlab and Mathematica. The nature of these environments make it possible for the user to invoke NetSolve in as simple or as complex a way as possible. The asynchronous interface further allows the user to make non-blocking request to NetSolve. The call returns immediately with a handle to the request that the user can use to probe to see if the request has completed and retrieve the results.

This interface allows users to invoke multiple calls to NetSolve that would then run on different hosts, further improving application turnaround time.

To aid in the development of SCAs that do Monte Carlo simulations, parameter sweeps and other applications with simple task-parallel structures, i.e. independently parallel programming, we have created a task farming interface. The task farming interface allows the user to make a single call to netsolve, requesting multiple instances of the same problem. Rather than single parameters, the user passes arrays of parameters as input/output and designates how NetSolve should iterate across the arrays for the task farm. The main challenge in this effort is scheduling. Indeed, for long running farming applications it is to be expected that the availability and workload of resources within the server pool will change dynamically. [14] discusses the design of this infrastructure and also presents an adaptive scheduling algorithm used by the task farming interface to assign tasks to the server resources.

• **Corollary 7:** The SCAI must be have multiple-language support and its components should be completely pluggable.

As already mentioned, the NetSolve API has been implemented in many programming environments. Further motivated by the need to support various platforms (among other things), we have implemented client proxies to act on behalf of the Netsolve client. The proxy, a separate process that resides on the client host, handles (almost) all interactions with the underlying meta-computing resources. With a standard interface between the client and all proxies, it is possible, especially for third party developers, to easily add new language support to the NetSolve system. They would simply write libraries that interface the NetSolve proxies from their language of choice, allowing programs of that language to become NetSolve-enabled. Figure 9 depicts the main idea behind the proxy. The client libraries interact with the proxy thanks to a standard API and the proxy interacts with the meta-computing system using system-specific mechanisms. The NetSolve proxy, for instance, uses the agent to discover services, contacts the appropriate server and establishes a session with that server who then receives input data from the client, executes his service and return output data.

• **Corollary 8:** The design of the SCAI must optimize performance with minimal overhead to the resources it occupies.

The NetSolve agent uses both static and dynamic information from the servers to assign requests to the best server at that point in time. The algorithms for each service is configured with a complexity that describes the computational time of the algorithm based on input sizes. Performance is measured by the LINPACK benchmark upon server initiation, and the server monitors its host reporting workload information. All these parameters, along with network bandwidth and latency information, are used whenever a request is received to rank the appropriate servers from best to worst. This list is sent

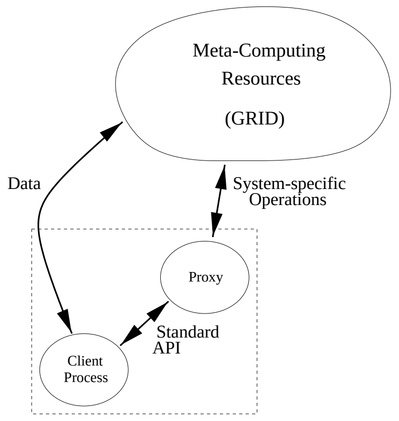

Figure 9: Proxy Architecture

to the client who then uses the fastest server to solve his problem in order to optimize performance.

The major drawback in distributed systems is often data transfers. Though not yet implemented, under consideration are heuristics to consider if performance might be improved by using the client's host as a server by uploading the necessary software once, rather than transferring data to a server (on possibly several occasions).

5 Conclusion

Scientific Computing Applications are a prominent part of the field of Computer Science. These applications and the computational scientists that implement them have unique characteristics that forge the software infrastructures needed to support large scale development. We have discussed the relevant characteristics they possess and present NetSolve, an environment for solving computational problems, as a system that addresses these issues. The system is a work-in-progress, and at the heart of this effort lies the philosophy that convenient interfaces and ease of administration are most important; every effort is made not to sacrifice these elements as the system evolves to meet the needs of the scientific computing community.

References

[1] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM : Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing.* The MIT Press Cambridge, Massachusetts, 1994.

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

[3] I. Foster and C. Kesselman, editors. *The Grid, Blueprint for a New computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.

[4] Distinctive Characteristics of Scientific Applications.

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[6] S. Balay, W. D. Gropp, and B. F. Smith. *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.

[7] R. Lehoucq, D. Sorensen, and C. Yang. *ARPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, first edition, 1998.

[8] B. Monien, R. Diekmann, R. Feldmann, R. Klasing, R. Luling, K. Menzel, T. Romke, and U. Schroeder. Efficient Use of Parallel & Distributed Systems: From Theory to Practice. In *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[9] R. Bramley, D. Gannon, T. Stuckey, J. Villacis, J. Balsabrumanian, E. Ackman, F. Breg, S. Diwan, and M. Govindaraju. Component Architectures for Distributed Scientific Problem Solving.

[10] D. Arnold, S. Blackford, and J. Dongarra. Users' Guide to NetSolve V1.3. Technical report, Computer Science Dept., University of Tennessee, May 2000.

[11] S. A Hutchinson, Shadid J. N., and Tuminaro R. S. Aztec user's guide: Version 1.1. Technical Report SAND95-1559, Sandia National Laboratories, 1995.

[12] D. C Arnold, D. Bachmann, and J. Dongarra. Request Sequencing: Optimizing Communication for the Grid. In *Euro-Par 2000 – Parallel Processing*, August 2000.

[13] D. C. Arnold, S. Blackford, J. Dongarra, V. Eijkhout, and T. Xu. Seamless Access to Adaptive Solver Algorithms. August 2000.

[14] H. Casanova, M. Kim, J. S. Plank, and J. Dongarra. Adaptive Scheduling for Task Farming with Grid Middleware. *The International Journal of Supercomputer Applications and High Performance Computing*, 1999. to appear.