Remote Access to Mathematical Software

Elizabeth Dolan, Paul Hovland, Jorge Moré,
Boyana Norris, and Barry Smith
Mathematics and Computer Science Division
Argonne National Laboratory
9700 S. Cass Avenue, Argonne, IL 60439-4844
[dolan,hovland,more,norris,bsmith]@mcs.anl.gov

Abstract

The network-oriented application services paradigm is becoming increasingly common for scientific computing. The popularity of this approach can be attributed to the numerous advantages to both user and developer provided by network-enabled mathematical software. The burden of installing and maintaining complex systems is lifted from the user, while enabling developers to provide frequent updates without disrupting service. Access to software with similar functionality can be unified under the same interface. Remote servers can utilize potentially more powerful computing resources than may be available locally. We discuss some of the application services developed by the Mathematics and Computer Science Division at Argonne National Laboratory, including the Network Enabled Optimization System (NEOS) Server and the Automatic Differentiation of C (ADIC) Server, as well as preliminary work on Web access to the Portable Extensible Toolkit for Scientific Computing (PETSc). We also provide a brief survey of related work.

1 Introduction

Network application services for business applications have become very popular in recent years. Systems that enable Internet access to scientific software have also emerged.

Several principal approaches to making software available over the network exist. One approach is to enable Web-based remote use of hardware and software resources in a fashion closely resembling local use. Users may need to have accounts on the remote machines. For example, the Grid Portal Toolkit (GridPort) [13, 19] provides access to a collection of services, scripts, and tools that allow NPACI users to run codes, access data, and communicate with NPACI's Globus-ready systems.

Other servers, such as the NEOS Server, may transfer the user's program or the problem specification and data from the user's machine to a remote machine, which then runs the code on the data and transfers back the result. The user does not necessarily need an account on the remote machine.

Another approach is to download the application from the server to the user's machine, where it operates on the user's data and generates the result locally. Finally, in a remote computing environment, only the user's data travels to the server, where programs based on numerical libraries operate on it and then return the result to the user. Some service providers, such as NetSolve [9], use this approach.

We have identified a number of issues in making our scientific software accessible on the Internet. Each of the three servers discussed in this paper addresses a subset of these issues.

- **User interface and problem representation.** A good user interface design is crucial to any network-enabled application. The benefit of providing Internet access to mathematical and scientific software would be diminished if the learning curve for using it remotely is too steep. Making existing software accessible over the network offers an opportunity for designing an interface which can serve a double purpose—hiding the complexity of scientific software and providing secure access to remote resources. In general, the user's input must be transformed to the format accepted by the mathematical software. If a server provides access to more than one type of software, as in the case of the NEOS

Server, providing a standard format for the problem representation makes it possible to extend the functionality of the server without having to modify its implementation.

- **Security.** Offering any type of service over the Internet exposes the software and hardware to malicious attacks. Internet-accessible software can potentially be used to gain access to protected system resources. On machines providing more than one service, a breach of one application can be used to disrupt the operations of another. We focus on security issues directly related to the servers discussed in this paper; although Web servers and browsers may be vulnerable to malicious attacks, we do not offer any general solutions for this type of problem.

  One possible general solution is to use an operating system such as Trusted Linux, HP Laboratiories' implementation of a secure version of Linux, which contains kernel-enforced controls [11]. In a trusted OS implementation, services and applications are run within separate compartments, and kernel-level mandatory checks ensure that processes from one compartment cannot interfere with processes from another compartment. Each compartment has a file system section associated with it and can access files only within that section. Network access is provided via narrow, kernel-controlled interfaces governed by compartment-specific rules specified by the system administrator. The idea is similar to that of Java security via a "sandbox"; however, while a secure kernel implementation controls the execution of all applications, the Java model relies on the application to determine and enforce its security policy. For example, when an applet runs inside the HotJava browser, HotJavaTM is the Java application that has determined the security policy for that applet.

  Other OS-based solutions focus on remedies for application-specific security vulnerabilities. One such approach is the system-call monitoring system (SMS) [8] developed collaboratively at Telcordia and SUNY. SMS augments the kernel's general-purpose implementation of system calls with an application-specific one, which incorporates exploitation detection and damage prevention mechanisms. While this approach eliminates reliance on software vendor security updates, it may cause significant performance degradation in some applications.

  In all three servers described in this paper, the issue of security is addressed by providing a narrow interface to the underlying software, without significantly reducing the functionality of the remote service. Each server implements additional measures, some of which are described in more detail in subsequent sections. While some security issues are common to most Internet application services, often there are unique challenges to providing secure access to mathematical software. In this paper we forego discussion on generic security issues and focus more on the application-specific details.

- **Distributed resource management.** While the Automatic Differentiation of C (ADIC) Server and Network-Enabled Optimization System (NEOS) Server provide access to software that usually runs on a single processor, the network-enabled servers themselves are distributed. The Portable, Extensible Toolkit for Scientific Computing (PETSc) is a parallel toolkit whose corresponding application server provides access to a distributed set of resources for each user request. A shared goal of all three servers is to provide good response times for user requests. Thus, some distributed resource management strategy is needed. We describe the approaches used in our server implementations in subsequent sections.

In the remainder of this paper we discuss the individual requirements, challenges, and implementation highlights of the ADIC, PETSc, and NEOS Servers.

2 The ADIC Server

The ADIC Server makes automatic differentiation (AD) available via the Web. Derivatives play an important role in a variety of scientific computing applications, including optimization, solution of nonlinear equations, sensitivity analysis, and nonlinear inverse problems. AD technology provides a mechanism for augmenting computer programs with statements for computing derivatives [15, 16]. In general, given a code C that computes a function $f : x \in \mathbf{R^n} \mapsto \mathbf{y} \in \mathbf{R^m}$ with n inputs and m outputs, an AD tool produces code C' that computes $f' = \partial y / \partial x$, or the derivatives of some of the outputs y with respect to some of the inputs x. In order to produce derivative computations automatically, AD tools systematically apply the chain rule of differential calculus at the elementary operator level.

ADIC is a source transformation tool for the automatic differentiation of ANSI C code [7, 17]. Source transformation AD tools extend the notion of a compiler by altering the functionality of the original program, augmenting it with derivative computations. Given a set of C source files, ADIC produces a new set of source code files augmented with derivative

computations. Limited C++ support is also available. The ADIC design allows easy expansion of its functionality through plug-in modules. A module specified at runtime interacts with the rest of the system via machine- and language-independent file interfaces. Language independence is achieved through the use of an intermediate representation, known as the AIF (Automatic differentiation Interface Form) [1, 18], which abstracts AD-relevant information from the more general language features. Although most modules target derivative computation by exploiting the chain rule, modules can be written to perform any language-independent transformation. Each module usually has a set of associated runtime libraries, which must be linked with the differentiated code.

The ADIC Server aims to provide an easy-to-use, highly accessible interface to ADIC and, potentially, other AD tools. Our goals include developing and implementing mechanisms for remote file management, fast response to user requests, scheduling of requests in a distributed environment, and assistance with tasks the user must perform after downloading differentiated files from the server. The URL for the ADIC Server is www.mcs.anl.gov/autodiff/adicserver.

We discuss the requirements of source transformation application server, including account and file management, user interface and server implementation. Users of the ADIC Server can upload source code written in ANSI-C, manage remote files, differentiate selected functions, and download code augmented with derivative computations. Using a simple driver and linking to the appropriate libraries, the user can then compile and run the differentiated code locally.

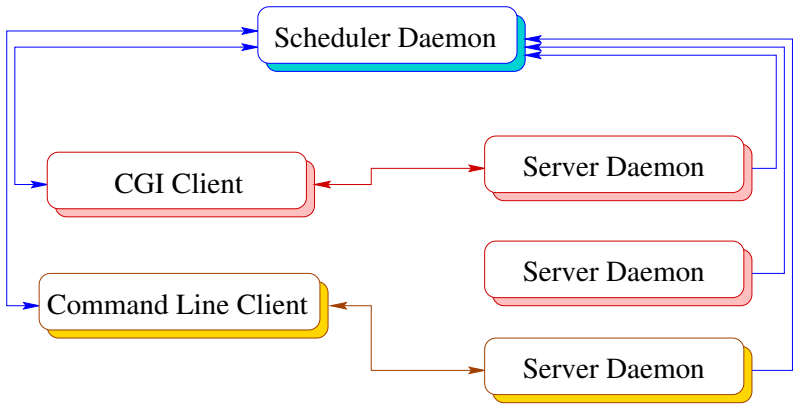
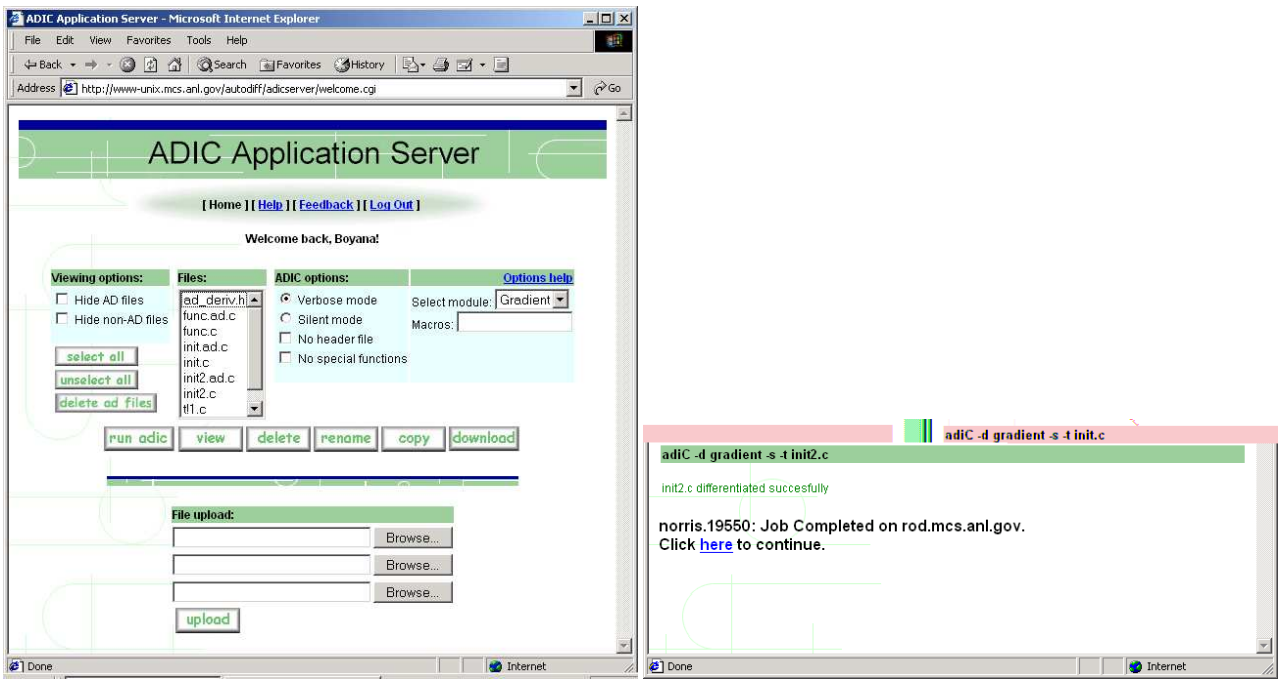


Figure 1: ADIC Server components.

Figure 1 illustrates the organization of the ADIC application server. The ADIC Server is the abstract entity that corresponds to a set of processes executing on different hosts, including one scheduler daemon, multiple server daemons, and multiple clients. All processes communicate using TCP/IP sockets. The main ADIC Server components are

- **Clients,** including a CGI-based Web client and a prototype command line version. The client handles user requests by first contacting the scheduler to obtain a server host name and port number. Then the client connects directly to the assigned server daemon and submits the request using a custom intermediate representation. Once the server begins fulfilling the request by applying ADIC to the files selected by the user, the client dynamically displays the server's output.
- **Scheduler daemon,** responsible for accepting job requests and selecting the (possibly) remote host on which to execute ADIC. The scheduler receives periodic updates from the server daemons, based on which the host with the smallest load (adjusted for the number of processors) is selected to service the client's request.
- **Server daemons,** responsible for receiving the user's request, parsing it, invoking ADIC, and transmitting the resulting code back to the client.

2.1 User Interface

The ADIC Server's Web interface is designed to enable a user to obtain derivative-enhanced versions of functions without being familiar with the AD process and the command-line interfaces of the tools. The main page contains virtually all

the options needed for uploading files, differentiating them, and downloading the resulting code. More advanced, or less frequently needed, functionality is included on separate Web pages.