

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 Laboratories' 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 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.

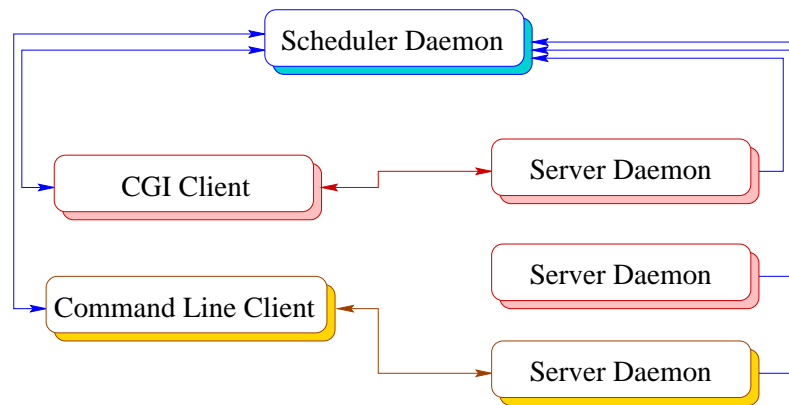


Figure 1: ADIC Server components.

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.

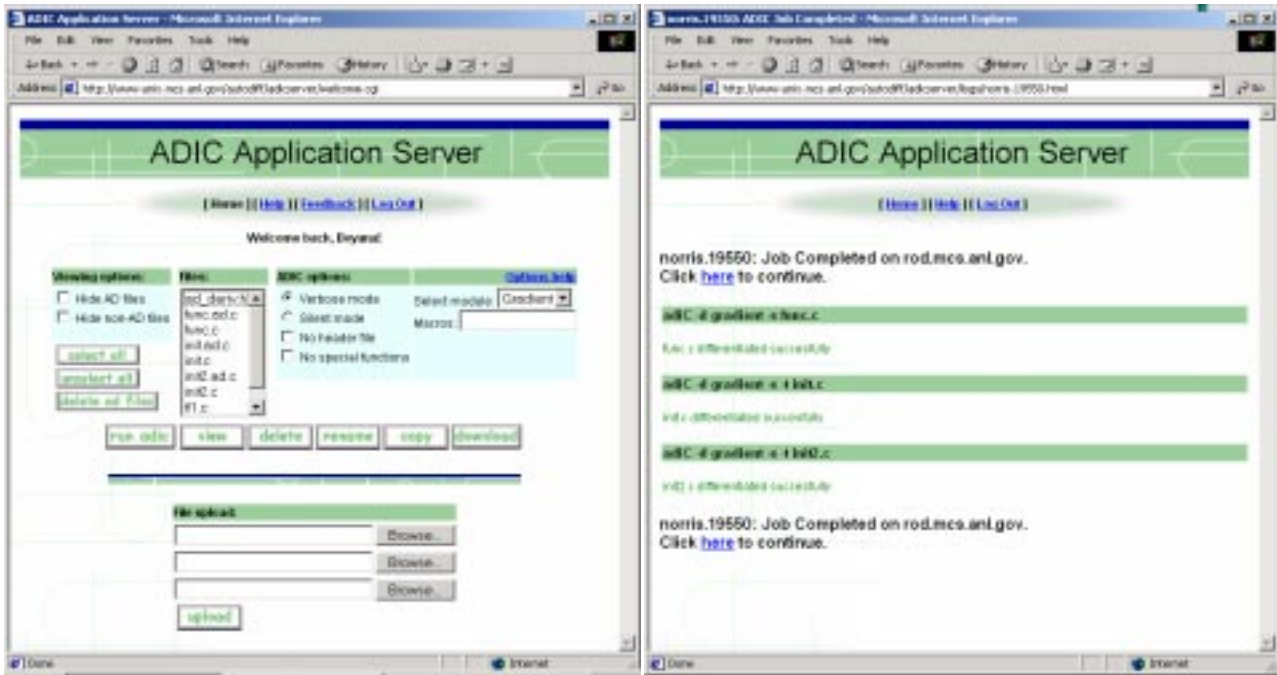


Figure 2: ADIC Server screen shots.

The following steps comprise a typical user session with the server:

- The new user registers and requests an account. A directory for remote storage of user files is created. This step is required only for new users. Existing users may be required to log in if a certain amount of time has elapsed since their last login. A cookie with some state information is created on the user's machine.
- The user uploads files to his or her account. The server imposes limits on the size of user files.
- The user selects files among the list of uploaded files, specifies ADIC options or uses the defaults, and directs the server to apply ADIC to them.
- The user views or downloads the ADIC-generated files containing derivative computations.
- At any time during the interactive session, the user can manipulate the files in the remote directory (delete, copy, rename, download).

During a session, the user can submit multiple requests for differentiating different sets of files. Before incorporating the differentiated code into an application, the user must follow the simple steps for downloading and unpacking the ADIC libraries, which are available for a number of Unix platforms. No additional software is required in order to use the code produced by the ADIC Server.

2.2 Security

Because the user code is processed by a C preprocessor, it is possible to gain access to information that can be used in a separate attack on the system providing the service. For example, source code uploaded by the user may expose the file, in which on some systems the encrypted root password is stored, by using the `#include ``/etc/passwd``` preprocessor directive. Then the user can use the server to process the file with ADIC, which would result in an error exit because the program is not syntactically correct once the include file has been inlined by a standard C preprocessor. ADIC stores the preprocessed source code in a temporary file, which is used in later steps of the processing. Temporary files are not removed

by ADIC in case of an error because they can be helpful in debugging. This is not normally a security concern; when used locally, the tool does not provide access to resources other than those ordinarily available to a user who is authorized to use the system. Without special precaution, however, when ADIC is used as a remote service, anybody can potentially gain access to resources they are not authorized to use, such as files containing the root password. To reduce this vulnerability, the ADIC Server makes sure the user has no access to the intermediate files produced by the tool.

Since the ADIC Server provides source transformation services, it may appear to be invulnerable to malicious attacks that rely on the execution of a piece of foreign code, such as buffer overflow. Applying ADIC to a file does involve running several independent executable files. Nevertheless, these executables never reside in the user's directory and are always invoked with the full path name. Furthermore, the server must always run in an environment whose executable search path does not contain the current directory (or the user's directory).

Names of files uploaded by the user may contain special characters, such as "|" (pipe), which on many systems is used to separate a sequence of commands, connecting the output of each command with the input of the next one. When such files are processed via ADIC, the file name appears on the command line executed by the server, which is usually running in a Unix shell environment. Special characters, such as "|" will be interpreted by the shell, changing the semantics of the command line, and executing a potentially harmful sequence of commands. Similarly, user-specified options containing special characters can also be interpreted by the shell as a sequence of executable statements instead of a single command applying ADIC to a source file. To prevent potential misuse of file names and command line options, the ADIC Server requires that all names and options be alphanumeric or include only a limited number of special characters that are not interpreted by the shell.

Another source of vulnerability is the use of cookies to maintain state during a session with the ADIC Server. Depending on its use, the information stored in cookies can make the server system vulnerable to attack by modifying the cookie or the browser itself. We avoid storing server information, such as paths to user accounts, in the user's cookie file.

The implementation of the ADIC Server contains a fair amount of JavaScript code. While this would not endanger the integrity of hardware or other applications running on the same machines, there is a potential for damaging the ADIC Server itself and the files of other users. At this stage, this potential vulnerability is not addressed by the ADIC Server.

When a locally installed copy of ADIC is used, the user can specify a script file, containing settings that control various aspects of the process of differentiating source code. This is a simple, yet powerful way to provide customization not covered by command-line options. However, some of the possible entries in such a control script present a significant security hazard when executed remotely. For example, instead of specifying a valid preprocessor executable (which overrides the default C preprocessor used by ADIC), the user can specify any other executable or shell command. No reliable way exists for scanning a control script and ensuring that it is not malicious. Therefore, the ADIC Server does not provide the option of uploading control scripts. Instead, we are working on an extension to the main interface, in which most options that would normally be a part of a control script are encapsulated in a Web form. The processing CGI script then would generate a control file guaranteed not to contain any executable overrides or other potentially dangerous elements.

2.3 Distributed Resource Management

The ADIC Server uses a simple scheduling strategy aimed at minimizing the response time for servicing user requests. The pool of available servers running on the network connect to a central scheduler and submit port and load information at regular intervals. The scheduler maintains a database of available servers and recent load information. When a client contacts the scheduler, the least-loaded server is selected, and its host name and port number are forwarded to the client. The client then connects directly to the assigned server and submits the problem parameters. The server transmits the output of executing the user's request to the client, with extra formatting for displaying on a Web page. When a server exits, it notifies the scheduler. If a server crashes, the scheduler detects the lack of recent load updates. In both cases, the host name is removed from the resource pool.

2.4 Future Work

The ADIC Server is currently under active development. In addition to extending the user interface with more advanced options, we are working on automating more of the steps involved in using AD tools. This includes providing the user with a driver template, which performs proper initialization and invokes the differentiated code (currently the user must do this manually).

We are also working on adding tools for remote administration. Web-based and command-line clients for job and file management control, server management and monitoring are being developed. We plan to add tools for gathering and viewing usage statistics.

3 The PETSc Server

PETSc [3, 4, 5] is a toolkit for the scalable, parallel numerical solution of partial differential equations. The software integrates a hierarchy of components that range from low-level distributed data structures for vectors and matrices through high-level linear, nonlinear, and timestepping solvers. The use of mathematical abstractions simplifies development by promoting code reuse and decoupling issues of parallelism from algorithm choices. Furthermore, well-defined abstract interfaces facilitate automatic generation of derivative code and development of an Internet-accessible PETSc Server.

A prototype server for compiling and executing sample PETSc applications has been developed. Three independent clients can be used to interact with a remote PETSc application. The server and all clients are implemented in Java and communicate via TCP/IP sockets. One advantage of using Java is that the clients can be executed in any browser that has the Java plug-in. The PETScRun client specifies the server machine, the application to compile, link, and run. Another client can be used to set PETSc runtime options dynamically as the application executes on the remote host. A third client provides access to the Alice Memory Snooper (AMS) [2], which is an application programming interface (API) and associated runtime system designed to help in writing computational steering, monitoring, and debugging tools. The AMS client allows the user to monitor and change the values of variables at runtime.

The PETSc Server is under active development. Future work on the PETSc Server involves extending the client interface to enable the user to submit a problem description to one or more PETSc components. For example, to solve a nonlinear PDE, the user may specify a function computing the solution on a local portion of the discretization. The server will compile the user function, produce its Jacobian using AD, and link the function and Jacobian with PETSc's nonlinear solver. The results of the parallel solution of the problem will then be transmitted back to the user. Since PETSc applications are parallel, scheduling user requests involves determining the best set of processors for each run, as opposed to determining the best uniprocessor for a sequential application.

The PETSc Server prototype relies primarily on Java's security and does not yet provide additional security measures. Currently no user code is uploaded or executed on the remote server. Ultimately, the PETScRun client will allow the user to upload a high-level problem description to the server. The security issues for the PETSc Server are similar to those of NEOS (discussed in Section 4.2), and will be handled in a similar manner. The current implementation of the Java clients requires that the user add the PETSc Server's Web address to their Java policy file. An alternative approach would be to use certificates, which are generally harder to implement.

4 The NEOS Server

The NEOS Server [10, 12] is an Internet-based client-server application that provides access to a number of optimization solvers, eliminating the need for downloading solver software, writing code to call the solver, or computing the derivatives for nonlinear problems.

The NEOS Server provides two main abstractions—that of data and that of services. Using the abstract implementation of services, the server can handle any user requests that conform to the standard representation. Although the NEOS Server is primarily used for optimization, its architecture would support any service that conforms to the standard data and service interfaces. The server assigns each job to a *solver* script that executes application software via requests to a communications daemon assigned to the application and solver station. The server can then delegate to the solver machines the tasks of interpreting data and executing software.

In the early days of NEOS, the server and solvers were all on the same system, so communication took place through files. Now, however, solvers are remote relative to the server, for both security and modularity. Hence, solvers must also have an Internet interface to the NEOS Server, just as users do. This interface is referred to as the NEOS Comms Tool, which downloads the user files from the NEOS Server, invokes the solver application, and returns the solver's results to NEOS.

The NEOS Comms Tool is a small client/server application whose client is the NEOS Server and whose server is a daemon running on each solver station. The NEOS Server, upon receipt of a job for a particular solver, connects to the machine running the Comms Tool daemon for that solver, uploads the user's data, and requests invocation of the remote solver. The Comms Tool daemon, upon receipt of this message, downloads the user's data and invokes the application

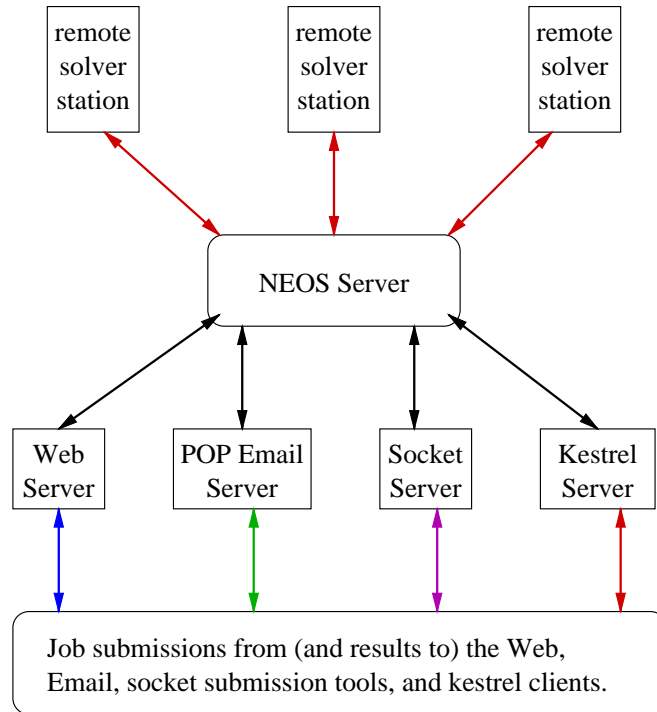


Figure 3: NEOS Server overview.

software. Intermediate job results can be streamed back over the connection to the server and from there, in turn, streamed back to the user. The Comms Tool daemon sends final results to the NEOS Server upon completion of the job, and the server formats and forwards the results to the user.

4.1 User Interface

The NEOS Server user interfaces currently consist of e-mail and Web submission tools (using TCP/IP sockets) and the CORBA-based Kestrel interface (see Fig. 3). Each interface provides the server with the user's data in a consistent manner. Given an input format and a list of solvers, the NEOS user submits a problem description consisting of the dimensions of the problem, the source code for the function evaluation, bounds and constraints routine (for constrained optimization), and the starting point.

The standard text representation of the problem parameters is interpreted by a NEOS solver, which compiles user-submitted functions, links them to the solver and corresponding driver, and remotely executes the resulting program. NEOS uses automatic differentiation to compute derivatives required by nonlinear solvers. In the case of Fortran user code, code for Jacobians or Hessians of nonlinear problems is generated by ADIFOR [6]. For C submissions, the AD tools used are ADOL-C [14] and ADIC.

Figure 4 illustrates the submission flow for three jobs arriving simultaneously from each of three user interfaces. After being sent in via e-mail, a Web form, or a submission tool form, the job submissions are located by the *receiver*. When it sees submissions, the *receiver* calls the *initializer.pl* script to add the submissions to the NEOS jobs directories. Then the *initializer* spawns *parsers* to decode the submissions. When the *parsers* complete their tasks, the *scheduler* is informed of the waiting jobs. The *scheduler* makes every effort to refer the jobs to the correct solver in the order they finished being parsed and spawns a *solver.pl* script for each job in turn. The *solver* scripts request job execution on the solver station determined by the *scheduler* via a *client* script written to handle socket communications with the remote NEOS Comms daemons. When the job is complete, results are sent back either through the helper scripts or by the *solver.pl* itself.

While the Web and socket interfaces display a similar flow of information, e-mail submissions require different amounts of attention at various stages. For example, Fig. 4 depicts how both the Web and socket submissions are presented to the

receiver script by NEOS helper scripts (CGI scripts and the *socket server* script). On the other hand, the *receiver* must actively request e-mail submissions from the e-mail server. The *parser* e-mails users that their e-mail interface submissions have arrived, and the *solver.pl* script e-mails results back to the user through an external mail program (MH) rather than relying on NEOS helper scripts. The *solver.pl* makes no effort to transmit intermediate results to the e-mail user as it does to the helper scripts; instead it waits until the job is complete to send results.

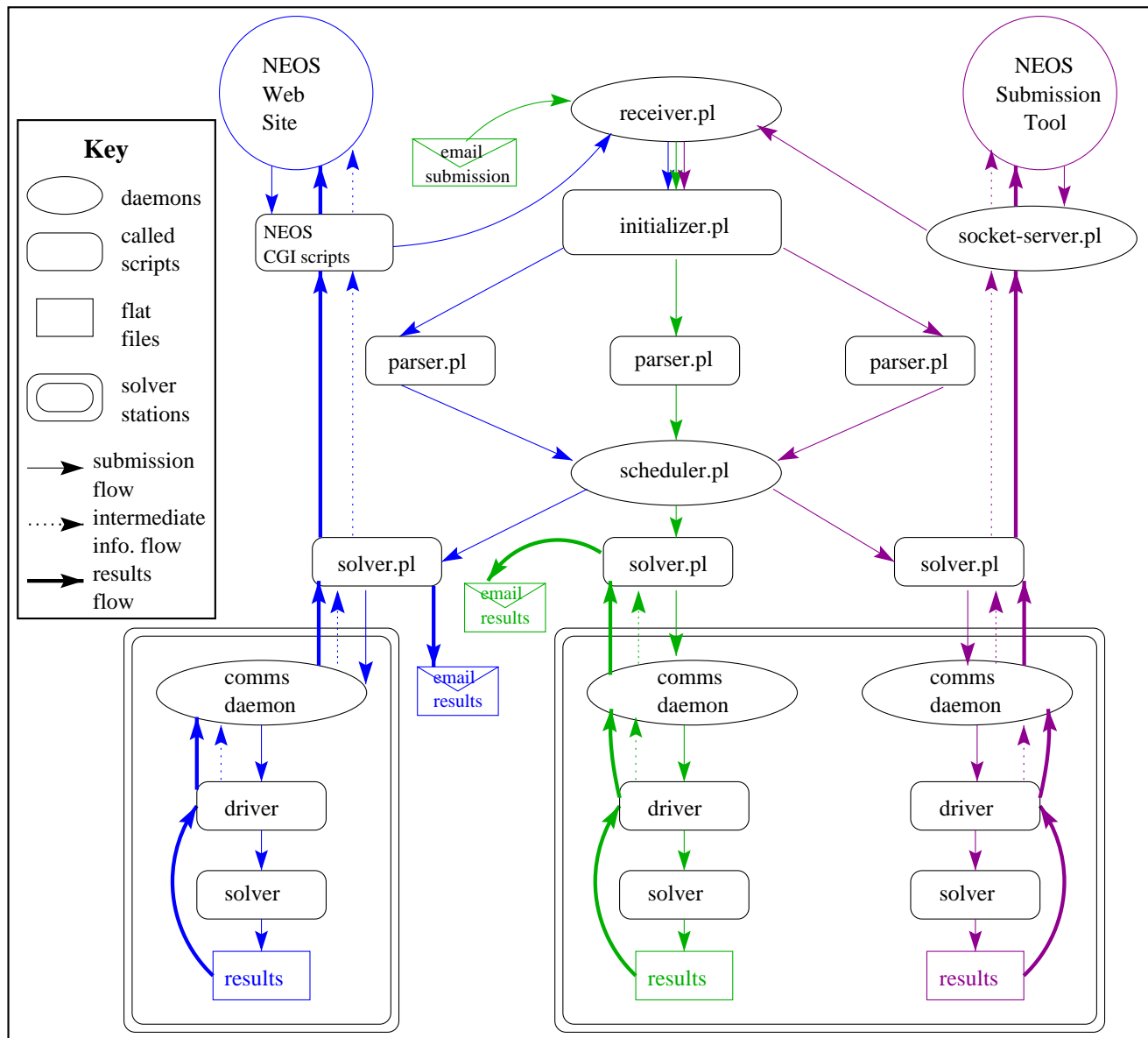


Figure 4: NEOS Server submission flow.

One distinction between the Web and socket submission interfaces is that the NEOS-created *socket server* daemon runs on the NEOS Server machine while the Web server daemon mentioned is the Web server (e.g., Apache) that hosts the NEOS Web site. The Web interface requires CGI scripts to facilitate communication with the NEOS Server via files while the *socket server* written for NEOS contains knowledge of the NEOS Server protocols internally.

4.2 Security

In this section we discuss the security issues specific to the NEOS Server. We also address areas of concern to remote solver administrators. Security was less of an issue in early NEOS versions, in which the server and solvers were on the same system. However, in the current, distributed version of NEOS, a number of security concerns must be addressed by the NEOS Server, the remote solvers, and the clients.

4.2.1 Server

The design of the NEOS Server allows new solvers to be added in a well-defined, straightforward manner [12]. The NEOS Server comes packaged with a suite of built-in administrative solvers, including ADMIN:ADDSOLVER, which is used by the solver administrator to add or remove a solver to the server.

4.2.2 Remote Solvers

The remote solver administrator is responsible for ensuring the security of the solver installation. The Comms Tool configuration can include disk usage limits (for submission files, as well as files created by the application), job time limits (if supported by the local operating system), maximum number of jobs, e-mail notification at job submission, and forwarding standard error output to the user.

In addition to Comms Tool options, the remote solver administrator can incorporate various security measures in the application driver. Prior to executing user code, the driver can scan the source code and corresponding object files to detect potentially malicious intent. For example, some solvers do not allow user code to include any read, write, or system commands. Security vulnerability can be reduced further by running the solvers with permissions that prevent access to protected resources or other applications.

Solvers based on problem solving environments (PSE), such as AMPL or GAMS, may need to address additional security concerns. The environment may need to be configured to differentiate between publicly available and private solvers. A previously private solver may be exposed inadvertently if the PSE software's configuration changes; for instance, when it is upgraded, the default paths to solvers may automatically change.

In solvers that allow binary user input, the file is read as data by the application, but not executed directly. This reduces the danger of runtime malicious attacks.

4.2.3 Clients

Each NEOS submission is assigned a unique ID and a password, which can subsequently be used for obtaining job status and results, ensuring that users cannot easily obtain (possibly sensitive or private) information on other user's submissions. Solver administrators have password protected access to all submissions on that solver.

4.3 Distributed Resource Management

This section contains a short overview of the principal components in the NEOS distributed environment. As illustrated in the example in Fig. 4, the NEOS Server consists of three multi-threaded daemons: receiver, Comms Tool, and scheduler. The receiver daemon checks for incoming jobs from any one of the interfaces in its spooling directory, and spawns processes to parse user submissions. The Comms Tool daemons coordinate communication between a (possibly remote) solver and the NEOS Server. The primary responsibilities of the Comms Tool are to accept connections from the server, download user data, execute the solver, and return the job results. The scheduler daemon is responsible for launching each job after the submission has been parsed by spawning a *solver.pl* script for each job. The *solver.pl* is then responsible for connecting to the NEOS Comms Tool daemon running on the remote solver machine. Jobs that cannot be scheduled on their requested solver are also handled by the *solver* script on the NEOS Server host machine, which will generally return job results explaining why the job could not be scheduled. The scheduler internally stores a list of all job numbers, their corresponding solvers, and current state (e.g., executing, queued for execution, scheduling error).

5 Conclusions

We have described the challenges in providing Internet access to three types of scientific software: a source transformation tool for automatic differentiation (ADIC), various types of optimization solvers (NEOS), and a toolkit for the parallel numerical solution of partial differential equations (PETSc). The servers described in this paper make the latest versions of research software under active development accessible to users with minimal installation and maintenance requirements. All three servers address a number of issues pertaining to their distributed implementation, such as resource management and security. However, the different requirements of each software domain are reflected in the implementation of the user interface, problem representation, scheduling strategy, and application-specific security provisions. Despite these differences, the NEOS Server has demonstrated that it is possible to develop a common infrastructure that can be used to streamline the process of providing Internet access to different types of scientific software.

Acknowledgements

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by the National Science Foundation (Challenges in Computational Science) grant CDA-9726385 and (Information Technology Research) grant CCR-0082807.

We thank Shannon Melfi of the University of Illinois at Urbana-Champaign for her substantial contribution to the initial design and implementation of the ADIC Server. We also thank Gail Pieper for proofreading an early draft of this manuscript.

References

- [1] AIF developer's page. www-unix.mcs.anl.gov/autodiff/AIF.
- [2] Ibrahima Ba, Christopher Malon, , and Barry Smith. AMS home page. <http://www.mcs.anl.gov/ams>, 2000.
- [3] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Lois Curfman McInnes, and Barry F. Smith. PETSc home page. <http://www.mcs.anl.gov/petsc>, 2001.
- [4] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.
- [5] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.0, Argonne National Laboratory, 2001.
- [6] Christian Bischof, Alan Carle, Peyvand Khademi, and Andrew Mauer. ADIFOR 2.0: Automatic differentiation of Fortran 77 programs. *IEEE Computational Science & Engineering*, 3(3):18–32, 1996.
- [7] Christian Bischof, Lucas Roh, and Andrew Mauer. ADIC — An extensible automatic differentiation tool for ANSI-C. *Software—Practice and Experience*, 27(12):1427–1456, 1997.
- [8] Thomas F. Bowen and Mark E. Segal. Remediation of application-specific security vulnerabilities at runtime. *IEEE Software*, 17(5):59–67, September/October 2000.
- [9] Henri Casanova and Jack Dongarra. Applying NetSolve's network-enabled server. *IEEE Computational Science and Engineering*, 5(3):57–67, July–September 1998.
- [10] Joseph Czyzyk, Michael P. Mesnier, and Jorge J. Moré. The NEOS Server. *IEEE Computational Science and Engineering*, 5(3):68–74, July–September 1998. Preprint ANL/MCS-P615-0996, Mathematics and Computer Science Division, Argonne National Laboratory.
- [11] Chris Dalto and Tse Huong Choo. An operating system approach to securing E-services. *Communications of the ACM*, 44(2):58–64, February 2001.

- [12] Liz Dolan. NEOS Server 4.0 administrative guide. Technical Report ANL/MCS-TM-25, Mathematics and Computer Science Division, Argonne National Laboratory, May 2001.
- [13] Gridport website. www.gridport.npaci.edu.
- [14] A. Griewank, D. Juedes, H. Mitev, J. Utke, O. Vogel, and A. Walther. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM TOMS*, 22(2):131–167, June 1996.
- [15] Andreas Griewank. On automatic differentiation. In *Mathematical Programming: Recent Developments and Applications*, pages 83–108, Amsterdam, 1989. Kluwer Academic Publishers.
- [16] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, 2000.
- [17] Paul D. Hovland and Boyana Norris. Users’ Guide to ADIC 1.1. Technical Memorandum ANL/MCS-TM-225, Argonne National Laboratory, 2001.
- [18] Lucas Roh, Paul D. Hovland, Jason Abate, and Christian Bischof. AIF component system. Unpublished.
- [19] Mary Thomas, Steve Mock, and Jay Boisseau. Development of toolkits for computational science portals: the NPACI HotPage. www.gridport.npaci.edu/pubs/publications/HPDC-9.pdf.