ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, IL 60439

ANL/MCS-TM-210

The Network-Enabled Optimization System Server

by

Michael Paul Mesnier

Mathematics and Computer Science Division

Technical Memorandum No. 210

August 1995

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

Contents

The Network-Enabled Optimization System Server

by

Michael Paul Mesnier

Abstract

Mathematical optimization is a technology under constant change and advancement, drawing upon the most efficient and accurate numerical methods to date. Further, these methods can be tailored for a specific application or generalized to accommodate a wider range of problems. This perpetual change creates an ever growing field, one that is often difficult to stay abreast of. Hence, the impetus behind the Network-Enabled Optimization System (NEOS) server, which aims to provide users, both novice and expert, with a guided tour through the expanding world of optimization. The NEOS server is responsible for bridging the gap between users and the optimization software they seek. More specifically, the NEOS server will accept optimization problems over the Internet and return a solution to the user either interactively or by e-mail. This paper discusses the current implementation of the server.

1 Introduction

The Network-Enabled Optimization System (NEOS) is being developed as part of the Optimization Technology Center, which is operated jointly by Argonne National Laboratory and Northwestern University. The goal of the NEOS server is simple: to direct users quickly to the proper optimization software and, from this software, to generate their solution. Users may communicate with the server in one of two ways: via e-mail or a Web interface. Essentially, then, the server acts as an interface between the users and the optimization software library.

The Network-Enabled Optimization System is composed as follows:

- Optimization software library - a library of numerical methods.
- NEOS server - job handling and interface to software library.
- NEOS Guide - a guide to optimization theory and practice.

The NEOS Server is implemented as a collection of Perl scripts, each of which has been appropriately named to reflect the function it performs. The main goals in implementing the server were speed, dependability, and extendibility. Hence, we sought to maximize the use of concurrency, extensive error-checking, and data

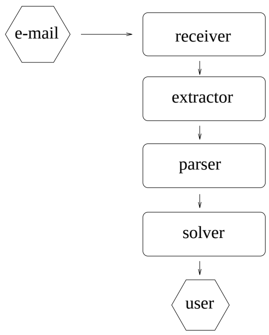

Figure 1: First server implementation

abstraction, respectively. Peripheral to the server are tools intended to enhance NEOS system administration.

Section 2 provides a detailed discussion of the server's components. Section 3 briefly describes the server interface to the Optimization Software Library. Section 4 focuses on the NEOS tools and their significance to the server.

2 The NEOS Server

The first implementation of the server consisted of only four scripts: the *receiver*, *extractor*, *parser*, and *solver*. With this implementation, there was no concurrency, and only minimal error-detection. Further, all processing was restricted to a single workstation. Despite the obvious limitations, this first implementation did provide us with a solid base to expand upon. See Figure 1.

The current implementation has improved on the prototype with the addition of concurrent execution, extensive error-detection, and the ability to use any Unix-based workstation on the Internet, regardless of architecture. Moreover, fault tolerance has been addressed, tools have been added for easier NEOS administration, and Web submissions are accommodated.

The server now consists of seven Perl scripts: *initializer*, *receiver*, *extractor*, *parser*, *scheduler*, *solver*, and *sender*. The server behaves as a pipeline. In other words, execution proceeds from the initializer through the sender without any exceptions. No script is skipped, and no script is executed more than once.[1] Peripheral to the server are two additional scripts: the *checker*, which ensures that the initializer is running after events such as a system crash or power failure; and the *cleaner*, which

[1] To be specific, no script is executed more than once per session. A session begins with the arrival of new job requests and the call from the receiver to the extractor (explained shortly).

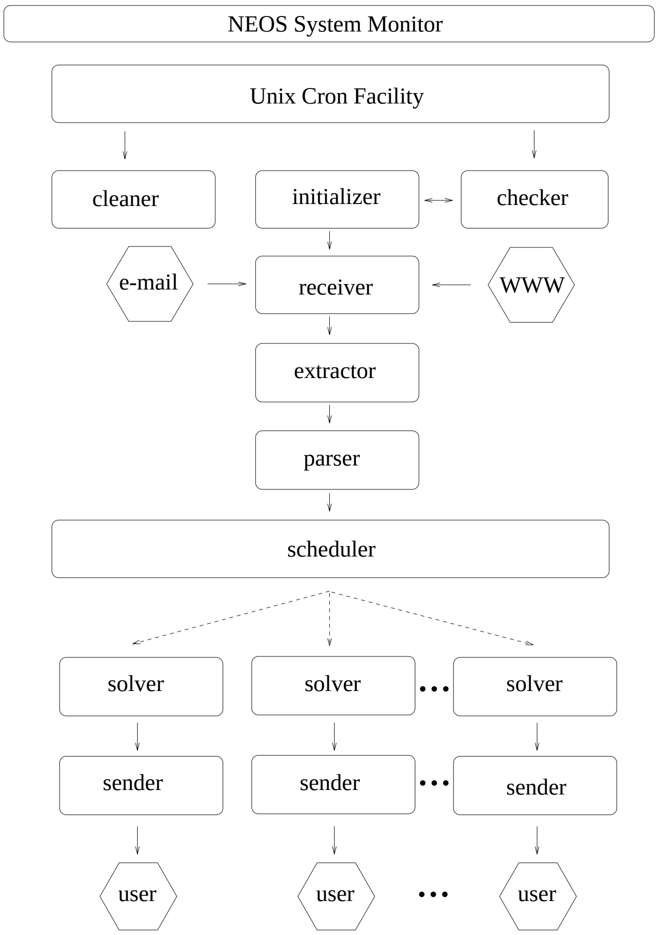

Figure 2: Current server implementation

performs routine garbage collection, archiving of previous job submissions, and log generation. See Figure 2.

2.1 The Initializer

Execution begins with the initializer daemon, whose primary purpose is to initialize all necessary environment variables and call the receiver to check for incoming jobs. Additionally, the initializer must communicate with the NEOS checker (described below). The NEOS directory structure is fully determined in the initializer via a collection of environment variables, each of which gives either an absolute or relative path to some point within the directory tree. This eliminates any problems incurred by a change in the file system. Because all subsequent scripts refer to the file system only through these environment variables, any changes in the file system will be

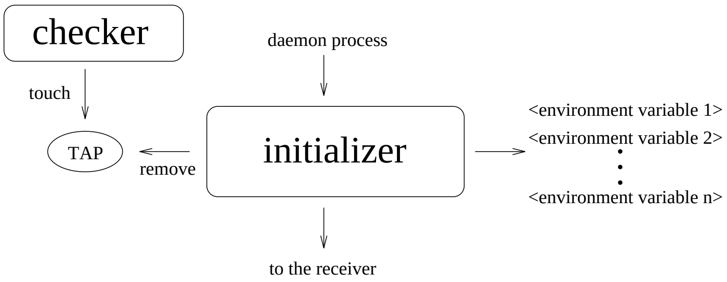

Figure 3: The initializer and checker

absorbed by the initializer.

The receiver is invoked every few seconds. The exact time is specified in a file located within the initializer's home directory. Because this file is read before each invocation of the receiver, any changes made to the file will be instantly realized. NEOS administrators may alter this file to adjust the frequency with which jobs are processed.

2.2 The Checker

Under perfect conditions, the initializer will run forever as a daemon process. We must, however, contend with system crashes and the like. To ensure such fault tolerance, the NEOS checker was added. As a companion to the server, the checker is responsible for checking the initializer and issuing a restart if the initializer is found dead or possibly in a catatonic state (e.g., inadvertently suspended).

The checker is implemented in an extremely simple fashion. Specifically, the checker issues the Unix "touch" command on a file in the initializer's directory. The initializer knows to remove this file when it appears; neglecting to remove the file is a direct signal to the checker that the initializer has crashed.

Because the checker is cron invoked, this implementation assumes that the Unix cron facility is crash resistant. Under this assumption, the NEOS system achieves a high level of fault tolerance. See Figure 3.

2.3 The Receiver

The call from the initializer to the receiver is simply a check for incoming job requests, from either e-mail or the Web. If jobs exist, control is passed on to the extractor; otherwise the pipeline is not entered, and control returns to the initializer.

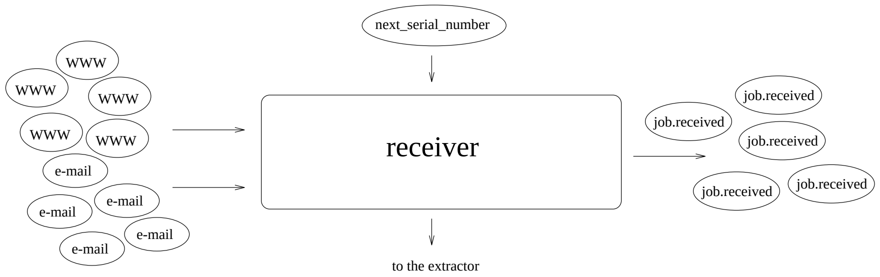

Figure 4: The receiver

As mentioned earlier, the extractor signals the beginning of the pipe.

NEOS is intended to process multiple jobs and hence operate in a concurrent fashion. This implementation requires each job to be uniquely identifiable from its neighbors. To address this issue, the receiver assigns incoming jobs a unique number, as well as a directory which serves as a temporary residence for the job as it is processed.

From Figure 2, one can see that concurrency is postponed until the scheduler is reached. Prior to this point, jobs are processed in small groups determined by the receiver. For example, if five jobs are waiting in the NEOS mailbox, the receiver assigns job numbers and creates directories for these five jobs. These five jobs are passed to the extractor, then to the parser, and finally to the scheduler. Section 2.6 describes the scheduling process and how NEOS achieves concurrent execution.

2.3.1 Submissions by e-mail

The NEOS system runs under a standard Unix account and thus collects e-mail from the account mailbox. However, entities other than NEOS clients may send mail to this account (e.g., root, MAILER-DAEMON). Hence, the receiver must maintain a list of usernames that will not be treated as clients. This also provides an easy method of prohibiting NEOS access to any users who have demonstrated imprudent behavior in the past.

2.3.2 Web Submissions

The Web interface to the server communicates directly with the receiver. Just as each e-mail message is spooled in the account mailbox, NEOS has a directory in which Web submissions accumulate, and it checks this directory for requests just as it does the mailbox.

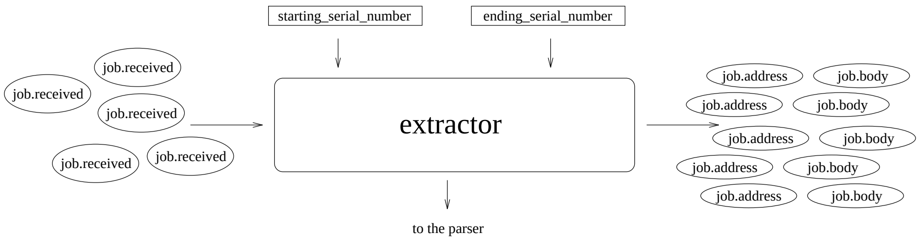

Figure 5: The extractor

Web submission did not appear until later in the server's development, and by this time the server was accustomed to handling submissions in e-mail format. In order to preserve the integrity of the server and adhere to the piping implementation, Web submissions appeared no differently from e-mail submissions. With exception of the receiver, Web submission is completely transparent to the server. See Figure 4.

2.4 The Extractor

The call to the extractor begins the breakdown of the job submission into its individual components. The extractor is concerned only with (1) the client's e-mail address and (2) the body of the mail message. As mentioned earlier, NEOS submission tool and Web submissions are identical to that of e-mail; hence, a "From:" field is manually appended to the top of all these submissions to make them appear as a regular e-mail. [2]

The first task of the extractor is to extract the return address from this "From:" field and place it into a file for later use by the NEOS sender. If the extractor fails to extract the return address, the job receives no further processing, yet still remains in the pipeline. Subsequent scripts are notified of this failure when the extractor issues the Unix "touch" command on a file within the job's directory (e.g., "touch ABORT"). Such an abortive feature allows the job to progress through the remainder of the pipe yet receive no operations.

Second, the extractor extracts the message body and places it in a file for later processing by the NEOS parser. Unlike the address, this extraction will never fail. At worst it will create an empty file if no message body exists. See Figure 5.

[2] This feature is actually handled by the NEOS and Web interfaces, not the NEOS server.

2.5 The Parser

Following a successful extraction, the parser further breaks apart the job submission by dividing the message body, referred to from here on as the job body, into its component parts. The only standard imposed here is that each component within the body be delimited by either one or two tokens. The following excerpt from an actual job submission illustrates the difference:

TYPE UM

BEGIN.COMMENT
THIS IS AN UNCONSTRAINED MINIMIZATION SUBMISSION.
END.COMMENT

The first token, "Type," delimits the information requesting the job type (i.e., the class of software in the optimization library). The second two tokens "Begin.comment" and "End.comment" delimit the text in between.

Each method in the optimization software library may tailor these tokens to whatever configuration serves them best. This information is specified in a file describing each token, the type of information it delimits, and the name of the file to place the information in. Other responsibilities imposed by the NEOS server on the individual methods within the software library are discussed in Section 3.

The parser needs only to know which method is being requested so that the proper token configuration file can be retrieved. This information is provided with a single token. For example, the type "UM" directs the parser to select the "UM" token configuration file and use this file to parse the remainder of the job body. If no "TYPE" token is found or an invalid method is requested, appropriate error messages will be returned to the user. Moreover, if a valid type is specified but invalid tokens (e.g., "begin.omment" instead of "begin.comment") are used, parsing errors are return to the user.

Any job that is unsuccessfully parsed will receive no further processing. Instead, the parser will issue an "ABORT" in the same manner as the extractor, signaling subsequent scripts to ignore the job and merely pass it through the pipe.

2.6 The Scheduler

The scheduler is the first of the scripts to use concurrency. Since certain optimization problems may run for hours, a concurrent approach was implemented to prevent bottlenecking. The scheduler is responsible for selecting a workstation to solve the problem on, and executing a separate NEOS solver for each of the jobs it receives.

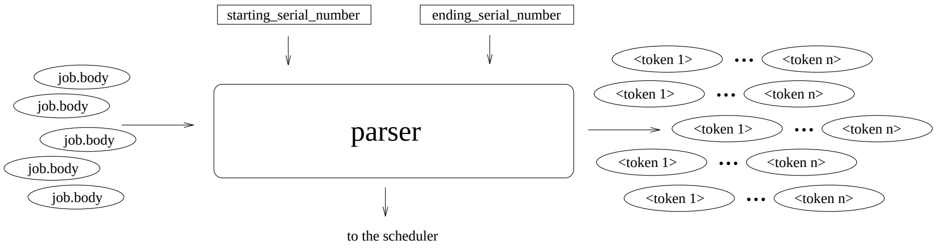

Figure 6: The parser

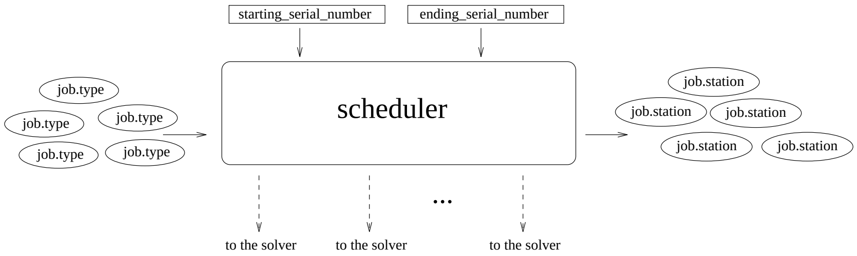

Figure 7: The scheduler

A feature recently added to the scheduler is the ability for each method within the software library to request exactly which workstations their jobs should be scheduled on. The methods can, however, accept the set of default workstations maintained by the scheduler.

Each job must wait in the scheduler until a workstation becomes available. We have imposed a "one job per workstation" limit to prevent any one workstation from becoming overworked. However, this limit be overridden by explicitly requesting a certain workstation in the job body. This feature is useful for software developers debugging their code; it should be disabled once the code is made publicly available. Not using the scheduling facilities of the server puts a workstations at risk of being overloaded with NEOS requests.

2.7 The Solver

The NEOS solver prepares each job for its venture into the optimization software library. As complicated as this may sound, the process is actually quite trivial because of the restriction that each method within the software library state to the server exactly *one* executable that is responsible for launching its code. The job of the solver is simply to call this executable and wait for its completion. Upon

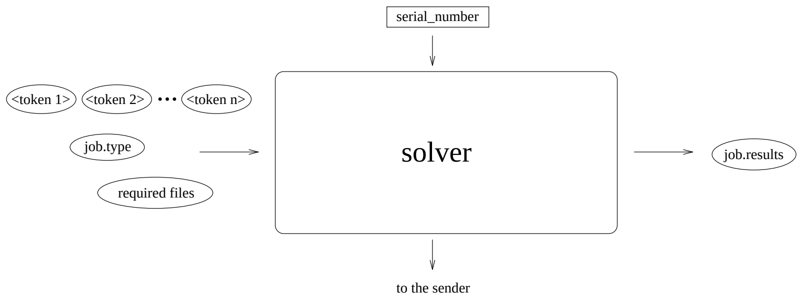

Figure 8: The solver

completion of the selected method, the solver retrieves the file containing the job results. To this file the solver then attaches a friendly NEOS header providing information pertinent to the server (e.g., time received, time completed, workstation selected).

The simplicity of this call to the software library may be deceiving. The executable is merely the front end to an often large set of algorithms. As far as the server is concerned, however, this call to the library does nothing more than produce a single file to which it attaches the NEOS header.

Until now, we have assumed all software within the optimization library to be running on the same file system. However, the NEOS solver can access software libraries on any file system and, hence, on any Unix-based network. This is the purpose behind the entire server: it simply acts as a liaison between the client and the optimization software, regardless of location. In brief, if methods do not reside on the same file system in which NEOS is running, they must specify their location (i.e., IP address) and instruct the solver to perform a file transfer. See Section 3 for a complete discussion.

2.8 The Sender

The sender is responsible for mailing job results back to the user whose address was previously extracted by the extractor. Further, because the sender marks the end of the pipeline, and thus the end of the NEOS process, a signal is sent to the cleaner to log and archive the job directory.

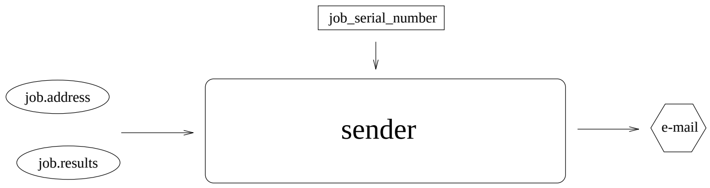

Figure 9: The sender

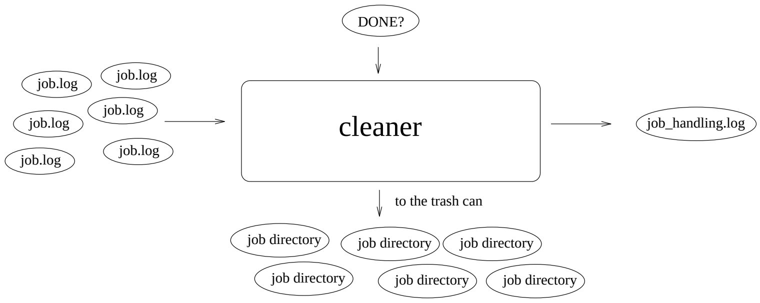

Figure 10: The cleaner

2.9 The Cleaner

The NEOS cleaner is responsible for archiving old job directories and creating a job-handling log. As each job was being processed, the individual scripts echoed their success or failure to a file within each of the local job directories. It is these files that the cleaner uses in log generation. The exact files maintained by the cleaner are located in the NEOS administration directory. All other files within the job directories are placed in an archive for future reference.

As mentioned earlier, the sender signals the cleaner when a job has been completed. The exact signal is once again the "touching" of a file (e.g., "DONE") by the sender. Until this file appears, the cleaner will not remove the directory. In addition, the cleaner will clean only if there exists a file in the cleaner's home directory (e.g., "CLEAN") that gives the cleaner permission to tidy things up. This implementation provides NEOS administrators with an easy method of enabling/disabling the cleaning process.

Just as the NEOS checker is called by the Unix cron daemon, so is the cleaner . See Figure 10.

3 Optimization Software Library

With a full understanding of the NEOS server in hand, let us now concentrate on the exact interface between the server and the optimization software library. Specifically, each method within the library must provide the following to the server:

- A unique solver name (e.g., Unconstrained Minimization = UM).
- The e-mail address of the software maintainer.
- The title of the method.
- An abstract of the method.
- The file that will serve as the token configuration file for the NEOS parser.
- The absolute path to the executable that will launch the code.
- The IP address of the remote workstation if the method resides on a different file system.
- An optional list of workstations if the method chooses to override the default set of workstations maintained by the scheduler.

All of the above files are to be placed in a "solvers" directory determined by the NEOS server, under a subdirectory with the unique solver name. For example, all Unconstrained Minimization files are located in the "solvers" directory under the subdirectory "UM." To activate a particular solver, the unique name must be appended to a "solver list" within this "solvers" directory. Once solver is activated, if any of the information listed above is missing, then the server must notify the method owner.

Here it becomes quite evident that the NEOS "solvers" directory essentially contains links to the software library, which may or may not reside on the local file system. No restrictions are imposed upon the developers of the software library. Furthermore, adhering to this protocol shields the software library developers from any changes in the NEOS server and, more important, frees them from worrying about the internal structure of the server.

4 NEOS Tools

In addition to the server and its peripherals, two additional tools have proved useful throughout the development of NEOS: the NEOS system monitor and the NEOS updating tool. These tools were developed as an aid for NEOS administrators.

4.1 NEOS System Monitor

Section 2.9 discussed how logs are generated, but did not mention the methods of access. Prior to the inception of the NEOS monitor, the NEOS staff had to find the directory containing the logs and carefully view the logs (i.e., make sure not to edit or delete the files). Granted, these are two simple steps, but two steps we believed to be unnecessary. Ideally, we wanted a friendly system monitor, which could capitalize on the amenities of an X-Window application.

Our answer was to develop a Tcl/Tk application, appropriately named the NEOS system monitor. Since two versions of NEOS are running continually, NEOS and Test-NEOS, this application offers log perusing from either system by way of a simple point and click. Each system currently offers the following logs:

- System run-time log: system messages pertinent to the execution of NEOS.
- Job-handling log: log of all submitted jobs.
- User log: log of all NEOS users.
- Cleaning log: account of the cleaner's whereabouts.
- Checker log: notification of system restarts.

4.2 Updating Tool

As mentioned, NEOS comprises two distinct systems: the test version and the publicly available version. All system development occurs in the test version where it receives a battery of tests prior to going online. Initially, updating the NEOS system was a trivial matter, since we managed only a few Perl scripts. Now, however, NEOS contains hundreds of files, making a system update quite complex.

To alleviate the pain of an update, we introduced the NEOS updating tool. Essentially, this tool maintains a list of files composing the entire NEOS application. This list must be maintained by the software developers. Updating then becomes the domain of the updating tool, which compares all files common to both NEOS and Test-NEOS, updating those that have been modified. If files within this managed file list exist in the Test-NEOS system only, the updating tool will ask for its inclusion in NEOS.

What was becoming an increasingly laborious process is now completely automated by the NEOS updating tool. Moreover, the hazards introduced from manual update (e.g., deleted files) have been eliminated.

5 Conclusion

We expect that the NEOS server will continue to enjoy future growth. NEOS was created to be expandable; hence, the power behind the NEOS server lies in its ability to efficiently manage an ever growing library of numerical software.

Optimization has long been a tool of mathematicians. Now, however, we are seeing a growing demand for optimization in both the private and the public sectors. The Network-Enabled Optimization System is designed to meet this demand.

Acknowledgments

The design of the NEOS Server was developed in consultation with Joe Czyzyk, Bill Gropp and Jorge Moré. Others contributing to this effort include Ali Bouaricha, Steve Wright, and Ciyou Zhu. The Web interface to the server was developed by Richard Marynowski. Many others have indirectly contributed to the NEOS Server; in particular, the developers of ADIFOR 2.0.