The Kestrel Interface to the NEOS Server*

Elizabeth D. Dolan†
Todd S. Munson‡

February 6, 2002

Abstract

The NEOS Server provides access to optimization solvers through the Internet with a suite of interfaces. In particular, the Kestrel interface enables the remote solution of optimization problems within the AMPL and GAMS modeling languages. Problem generation, including the run-time detection of syntax errors, occurs on the local machine using any available modeling language facilities. Solution takes place on a remote machine, with the result returned in the native modeling language format for further processing. No significant differences exist between local and remote solutions. A byproduct of the Kestrel interface is the ability to solve in parallel multiple problems generated by a modeling language.

1 Introduction

The NEOS Server [2, 3] is a convenient gateway to optimization software and services on the Internet. Interested parties can evaluate many different packages for solving their particular optimization problems without installing the software on their local machine. Instead, the user communicates a problem to the NEOS Server through e-mail, the Web, or a socket-based graphical user interface. When using these interfaces, the local machine is responsible for submitting a representation of the problem, for example, source code or a model written in modeling language syntax, to the NEOS Server and waiting for the result. Problem validation and solution happen on remote resources.

An alternative for individuals with local access to the AMPL [5] or GAMS [1] modeling languages is to use the Kestrel interface to the NEOS Server. In

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing, 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.

†Electrical and Computer Engineering Department, Northwestern University, and Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439; e-mail: dolan@mcs.anl.gov

‡Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439; e-mail: tmunson@mcs.anl.gov

this case, a problem is generated using any of the available modeling language facilities on the local machine, and the NEOS Server is used only for remote solution. Because the problem is generated on the local machine, users can access the file-system and other utilities when specifying their model; and all syntax errors are detected when the internal model representation is generated by AMPL or GAMS. Furthermore, the results returned through the NEOS Server are available in the native modeling language format for further processing.

No significant differences exist between local and remote solutions when using the Kestrel interface. Whenever a **solve** command is initiated in either AMPL or GAMS, the modeling software generates an internal representation of the current problem and calls a corresponding local solver. When using the Kestrel interface, the local solver executed is a Kestrel client, which contacts the NEOS Server, submits the generated model representation within tags understood by NEOS, and waits for the results. When the Kestrel client exits, the results are read by the modeling language as if the solve were performed locally. A byproduct of the Kestrel interface is the ability to easily solve multiple models in parallel [4].

The concepts employed when using the Kestrel client are the same for both the AMPL and GAMS interfaces, but the mechanics of the implementations differ because of the nature of the modeling languages. Documentation for each interface follows, along with a discussion of the technical details.

2 AMPL Interface

Two methods exist for using the Kestrel interface in AMPL. The first method simply replaces the normal solver used during a **solve** command with the Kestrel "solver," which submits the current problem to the NEOS Server and retrieves the results. The second method uses separate submit and retrieve facilities, which can be used to submit multiple problems to the NEOS Server before retrieving any of the results. In both cases, the user specifies a remote solver as one of the options to the Kestrel client.

Users can download the Kestrel client for many different architectures from the NEOS Server Web site. To install the executable, users should unzip the archive in a directory within their path, which enables the AMPL interpreter to locate the client during the **solve** command. Also contained in the archive are three command scripts used for the submission, retrieval, and kill capabilities discussed in the sequel.

Once the software is installed, the Kestrel interface can be used to solve an optimization problem remotely. For example, consider the original code using LOQO [8] to solve an optimization problem on the local machine.

```
ampl: model steel.mod;
ampl: data steel.dat;
ampl: option solver loqo;
ampl: option loqo_options 'outlev=2';
ampl: solve;
```

The corresponding code to solve the same problem remotely through the NEOS server follows.

```
ampl: model steel.mod;
ampl: data steel.dat;
ampl: option solver kestrel;
ampl: option kestrel_options 'solver=loqo';
ampl: option loqo_options 'outlev=2';
ampl: solve;
```

The two differences are that the solver is changed to **kestrel**, which is the client responsible for submission and retrieval; and the remote solver to be used is identified with the `kestrel_options` solver options, which set the `solver` to `loqo` in this instance. Any remote solver options are set with the appropriate `solver_options`, which in this case would be `loqo_options`. If the remote solver requires the use of AMPL auxiliary files, use the option `kestrel_auxfiles` to specify which files AMPL should create, as discussed in the AMPL documentation. The kestrel client automatically assigns the value of `kestrel_auxfiles` to the appropriate `solver_auxfiles` option for reading by the remote solver.

After the problem has been submitted to the NEOS Server by the Kestrel client, information is written to the console indicating the job number and password assigned by the NEOS Server for the particular solve. The output also indicates a Web site that can be used to monitor the progress of the solve, for example,

```
Job has been submitted to Kestrel
Kestrel/NEOS Job number    : 1234
Kestrel/NEOS Job password  : abcd
Check the following URL for progress report :
   http://www-neos.mcs.anl.gov/neos/neos-cgi/check-status.cgi
```

The use of network communication increases the likelihood that a particular solve will terminate abnormally, for example, if the connection to the network is lost. If this happens, the job number and password reported can be used to access the job when the system comes back on-line. For example, we can communicate the above job and password to the Kestrel client with the `job` and `password` solver options.

```
ampl: model steel.mod;
ampl: data steel.dat;
ampl: option solver kestrel;
ampl: option kestrel_options 'job=1234 password=abcd';
ampl: solve;
```

If the job and password solver options are set, the **solve** command waits for and reports the results of the corresponding NEOS job.

The job number and password information also enable a user to submit a job and at some later time retrieve the results. Currently, the NEOS Server keeps these jobs for three days after their completion before removing them from the system. To continue other modeling language processing, the user can interrupt the Kestrel solve manually and retrieve results later by setting the `job` and `password` options appropriately. A better alternative, however, is to use the commands scripts for separate submission and retrieval.

The submission and retrieval scripts are invoked in AMPL by using the **commands** facility. By default, AMPL accesses only those scripts that are in the directory in which the AMPL interpreter was invoked. Therefore, to use the provided commands, the user first must copy the scripts to the current working directory. Separate submission is achieved by replacing a solve with the **kestrelsub** and **kestrelret** pair of commands:

```
ampl: model steel.mod;
ampl: data steel.dat;
ampl: option solver kestrel;
ampl: option kestrel_options 'solver=loqo';
ampl: commands kestrelsub;
ampl: commands kestrelret;
```

The **kestrelsub** command prepares the current problem for submission and sends it to the NEOS Server. The NEOS job number and password are then reported. The **kestrelret** command retrieves the results. Any models submitted with **kestrelsub** should be retrieved with **kestrelret**.

The separate submission and retrieval capability allows a user to perform simple parallel processing within AMPL. Kestrel submissions and local solves can be performed before retrieving the results from a **kestrelsub** command. For simplicity, the retrieves are performed in the order in which the jobs were submitted. The form of this approach is as follows:

```
ampl: model steel.mod;
ampl: data steel.dat;
ampl: option solver kestrel;
ampl: option kestrel_options 'solver=loqo';
ampl: commands kestrelsub;
ampl: let steelscalar := 5.0;
ampl: commands kestrelsub;
ampl: commands kestrelret;
ampl: commands kestrelret;
```

More sophisticated sequences are possible. For example, the user could solve some of the models locally or use the **problem** statement to submit different models.

Finally, the user has the ability to kill submitted jobs from within AMPL. When a Kestrel solve is manually interrupted, the job normally continues running on the remote solution machine assigned by the NEOS Server. These resources can be freed by sending a kill request for the remote job. Depending on the solver and remote system, terminating the job through the NEOS Server may not be possible, but attempting to do so is simple. The user sets the job number and password in the `kestrel_options` and calls the **kestrelkill** command as in the following.

```
ampl: option kestrel_options 'job=1234 password=abcd';
ampl: commands kestrelkill;
```

Attempts to obtain results from a killed job would likely lead to a solution file unintelligible to AMPL.

3 GAMS Interface

The Kestrel interface to the NEOS Server for GAMS is similar to the one written for AMPL. The installation process involves placing the Kestrel archive for a particular architecture into the GAMS system directory and using the **gamsinst** program to unzip and install the Kestrel-related "solvers."

After successful installation of the Kestrel package, the **kestrel** solver can be used to solve a GAMS model remotely. For example, consider the `trnsport` model from GAMSLIB [1]. It can be solved locally in GAMS through the following statements,

```
model trnsport /all/;
solve trnsport using lp minimizing z;
```

which specify the `trnsport` model and solve it with the default linear programming solver. We can add an `option` statement to the code to explicitly specify the solver. For example, if we change the linear programming solver to MINOS [6], the code becomes

```
model trnsport /all/;
option lp = minos;
solve trnsport using lp minimizing z;
```

To solve the same problem remotely through the NEOS Server, we simply change the linear programming solver to `kestrel`.

```
model trnsport /all/;
trnsport.optfile = 1;
option lp = kestrel;
solve trnsport using lp minimizing z;
```

The statement `trnsport.optfile = 1` specifies that an options file, called `kestrel.opt`, will be used. The options file contains the remote solver name as well as any options for the remote solver. In particular, to use MINOS as the remote solver, we would write the following `kestrel.opt` file:

```
kestrel_solver minos
```

A subsequent run of the code through the GAMS interpreter results in the `trnsport` model being solved through the NEOS Server with the MINOS solver.

As with the AMPL interface, once the job is submitted to the NEOS Server, a job number, password, and Web address are displayed to the screen, which provide information on accessing the job and viewing the intermediate output, for example,

```
Job has been submitted to Kestrel
Kestrel/NEOS Job number    : 1234
Kestrel/NEOS Job password  : abcd
Check the following URL for progress report :
   http://www-neos.mcs.anl.gov/neos/neos-cgi/check-status.cgi
```

If the NEOS Server or the network becomes unavailable after the submission, a particular job can be retrieved by setting both the `kestrel_job` and `kestrel_password` in the options file.

```
kestrel_solver minos
kestrel_job 1234
kestrel_password abcd
```

Re-issuing the command **gams trnsport** with this options file will retrieve the results for the specified job number.

Separate submission and retrieval can also be issued by using the `kestrelsub` and `kestrelret` solvers, respectively. The GAMS convention is to name the options file `solver.opt`, where `solver` is the name of the solver used. With the submit and retrieve commands, we break with this convention and use `kestrel.opt` for the options file, instead of the expected `kestrelsub.opt` and `kestrelret.opt`. Therefore, to solve the `trnsport` model with the separate submission and retrieval facilities, we would write the following code:

```
model trnsport /all/;
trnsport.optfile = 1;

option lp = kestrelsub;
solve trnsport using lp minimizing z;

option lp = kestrelret;
solve trnsport using lp minimizing z;
```

with the `kestrel.opt` file containing the relevant `kestrel_solver` option.

The submit and retrieve facilities enable simple parallel processing capabilities within GAMS. Any number of submission and solves (including remote solves) can be performed before retrieving any results. For simplicity, we assume a work queue model in which the jobs are retrieved in the order submitted. Furthermore, the submit and retrieve ignore any job and password information in the options file.

Finally, GAMS also has a kill facility implemented by using the `kestrelkill` solver. In order to use this facility, a model must be present so that the solver can be invoked.

```
model trnsport /all/;
trnsport.optfile = 1;
option lp = kestrelkill;
solve trnsport using lp minimizing z;
```

The `kestrel.opt` file in this case should contain the job number and password of the job to kill. Subsequent attempts to obtain the results from a killed job should be avoided if possible because results will likely be mangled.

4 Technical Details

The Kestrel clients for AMPL and GAMS are written in C++ with all of the communication between client and server performed by using the CORBA specification [7]. This interface to the NEOS Server is possible because of the behavior of the AMPL and GAMS modeling languages when a "solve" command is encountered. Three steps are taken.

1. An internal representation of the current problem is written to disk.
2. The desired solver is located and executed with appropriate command line options, and the solver writes a solution file.
3. Finally, the solution file produced by the solver executable is read by the interpreter, which resumes processing.

The Kestrel client is a replacement for the local solver that relays the appropriate intermediate files to the NEOS Server in NEOS token-delimited submission format and obtains the results, which are then written to the correct solution file. When results are requested from the Kestrel client, we simply wait for the appropriate results to become available and write the solution file.

Special processing of the GAMS control file, `gamscntr.scr`, is performed by the Kestrel client code. The control file contains all the information for the problem and is located in the scratch directory. This file is parsed by the Kestrel client to replace the absolute file paths with relative file paths, and all information about the client GAMS installation, including the license information, is removed before sending the job to the NEOS Server. The NEOS license for GAMS is patched into the control file on the server side.

The separate submission and retrieval commands maintain a work queue. For both AMPL and GAMS, the work queue is a file containing a listing of the submitted job numbers, passwords, and remote solver names for jobs that have not been retrieved. The job number, password, and solver information is appended to the work queue file for each **kestrelsub**, and the first entry is removed from the work queue during each **kestrelret**. The **kestrelret** command removes the work queue file when it becomes empty.

The AMPL interface writes the work queue to a file created based on the process identification and the TMPDIR variable. For example, if the process identification is "1234" and the TMPDIR is "/tmp/", then the work queue file will be located in a file called "/tmp/at1234.jobs". The location and name of the file are similar to those used by AMPL for temporary NL and SOL files. This location can be affected by changing the TMPDIR variable. Furthermore, since the submission and retrieval are performed by using **commands**, as opposed to a **solve**, the submission script manually writes the current problem's description to a `kestrel.nl` file contained in the current directory, and the retrieve writes the solution to a `kestrel.sol` file. Unfortunately, the `kestrel.nl` and `kestrel.sol` are not removed when the AMPL session ends, and the user should remove them manually.

For completeness the `kestrelsub` command does the following:

```
option ampl_id (_pid);
write bkestrel;
shell 'kestrel submit kestrel';
```

where the first command saves the process identification into a variable accessible by the kestrel client, the second manually writes the current model to disk, and the last submits the problem to the NEOS Server. The `kestrelret` script is similar:

```
option ampl_id (_pid);
shell 'kestrel retrieve kestrel';
solution kestrel.sol;
```

where the **shell** command retrieves the solution file, and the **solution** command forces the AMPL interpreter to read the solution file. The `kestrelkill` is implemented with the single command,

```
shell 'kestrel kill kestrel';
```

The GAMS interface writes the work queue to a `kestrel.scr` file contained in the scratch directory of the current GAMS process. The scratch directory is automatically removed when the GAMS process exits, unless explicitly kept by the user with the **gamskeep** routine.

5 Conclusion

The Kestrel interface augments those interfaces currently available by NEOS and offers many advantages. The main advantage is that all models are created on the local machine, enabling users to debug their models locally and use any of the modeling language mechanisms when specifying their model. Another benefit is that the results are made available within the modeling language, which means that the users do not have to parse a results text file to use the answers generated. Moreover, the interface allows users to implement simple parallel programs.

Acknowledgments

We thank Bob Fourer for his assistance in testing the code in the early stages of development.

References

[1] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, 1988.

[2] J. Czyzyk, M. P. Mesnier, and J. J. Moré. The NEOS server. *IEEE Journal on Computational Science and Engineering*, 5:68–75, 1998.

[3] M. C. Ferris, M. P. Mesnier, and J. Moré. NEOS and Condor: Solving nonlinear optimization problems over the Internet. *ACM Transactions on Mathematical Software*, 26:1–18, 2000.

[4] M. C. Ferris and T. S. Munson. Modeling languages and Condor: Metacomputing for optimization. *Mathematical Programming*, 88:487–506, 2000.

[5] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 1993.

[6] B. A. Murtagh and M. A. Saunders. MINOS 5.0 user's guide. Technical Report SOL 83.20, Stanford University, Stanford, California, 1983.

[7] J. Siegel. *CORBA - Fundamentals and Programming*. John Wiley & Sons, New York, 1996.

[8] R. J. Vanderbei. LOQO user's manual – Version 3.10. *Optimization Methods and Software*, 12:485–514, 1999.