Running BeoPEST

Willem A. Schreüder
willem@prinmath.com

Introduction

BeoPEST is a special version of Parallel PEST inspired by Beowulf Clusters. A Beowulf Cluster is simply a number of commodity computers that are tied together using a reasonably fast network. The name is derived from the original Beowulf computer built by Thomas Sterling and Donald Becker at NASA. Although strictly speaking a Beowulf Cluster runs a common operating system (typically Linux), the term is used much more loosely here. In fact, BeoPEST is actually designed to form an ad hoc cluster on the fly. The cluster can comprise any set of computers than are able to communicate via the internet.

BeoPEST differs from the the traditional Parallel PEST only in how it communicates with slaves and how slaves know what to do. In the traditional Parallel PEST, a master process creates files using a template and read the results using the instruction file, in addition to performing the actual parameter estimation calculations. The slave simply executes the model runs. In BeoPEST, the master process still performs the parameter estimation calculations exactly as before, but instead of writing and reading files, it sends the set of parameters to be run to the slave, and receives observations from the slave in binary form over a network connection. The BeoPEST slave is smart and creates the model input files from the parameters as instructed by the master, runs the model, extracts the observations from the model output files, and sends the resulting observations back to the master. Therefore as much of the work as is possible is offloaded to the slave, and the master only deals with the parameter estimation proper.

BeoPEST comes in two flavors, BeoPEST/TCP and BeoPEST/MPI. BeoPEST/TCP uses TCP/IP connections to form an ad hoc cluster consisting of any set of computers that can communicate via a local area network or the internet. BeoPEST/MPI uses the Message Passing Interface (MPI) protocol to communicate and is more suited to running BeoPEST on a tightly integrated cluster. BeoPEST provides a single **ppest** executable, and switches between the traditional file based parallel pest, BeoPEST/TCP and BeoPEST/MPI using command line flags.

The BeoPEST implementation also has a number of cutting edge features for high performance computing. BeoPEST/TCP is distributed in the sense of grid computing. In

particular, this means that the slaves does not need to be on the same local area network as the master. BeoPEST/TCP allows some of the slaves to be at the other side of an internet connection. Thus, BeoPEST/TCP can tie multiple machines or even multiple clusters of machines together.

BeoPEST scales well. BeoPEST can be run efficiently with hundreds or thousands of slaves serving a single master. The inherent granularity of the problem may restrict the number of slaves that can be used efficiently, but BeoPEST can generally achieve high efficiencies.

BeoPEST/TCP is fault tolerant. It gracefully recovers from failed nodes. In addition, new nodes can be dynamically added and removed from the slave pool.

BeoPEST/TCP supports heterogeneous environments. The slave pool can consist of a mix of operating systems and hardware. For example, some slaves may be run on Windows workstations, while others may be run on a Linux servers using Intel hardware, while yet other slaves run AIX on IBM Power hardware. BeoPEST/TCP will readily accommodate such a heterogeneous environment.

BeoPEST does not need any additional files. Traditional Parallel PEST requires a Run Management File (RMF). BeoPEST only requires the Pest Control File (PST). All the information required by both the master and all slaves is contained in the PST file or supplied on the command line.

Running BeoPEST/TCP

With traditional Parallel PEST, the slaves are started first, and the master last. With BeoPEST/TCP, the master is started first, and slaves are started as needed.

The BeoPEST/TCP and traditional Parallel PEST executable programs are actually the same. To start the program in BeoPEST/TCP mode, the **ppest** program is started with the **/H** command line option. The **/H** option takes a port number which is preceded by a colon. The port number can be any unused port on the host where the master is run. Port numbers are positive integers between 1024 and 65536. Typically a port number such as 4004 is unused, so the master is started as

ppest foo /H :4004

where foo is the name of the parameter estimation. The master will read **foo.pst** to determine the parameters, observations, pest control parameters and the like. However, the master will not run the model. Instead it will wait indefinitely for one or more slaves to contact it. The port

number specifies the port on which the slaves should contact the master. Note that if the **/H** flag is not specified, the program will revert to the traditional Parallel PEST behavior as specified in the run management file.

The **pslave** program from traditional Parallel PEST does not exist for BeoPEST. Instead, the **ppest** program is used again using the **/H** flag. If the master is run on a computer *masterhost*, the slave is started as

ppest foo /H masterhost:4004

Note that the PST file must be exactly the same as the PST file read by the master and the port number (4004 in this example) must be the same as the master. The presence of a host name before the port number distinguishes the slaves from the master. The slave will read the PST file to get the names of all the parameters and observations. The slave will also determine the template, instruction and model files from the the PST file. This is sufficient information for the slave to be able to create model input files based on parameter values supplied by the master and run the model. Once the model run completes, the slave can extract the observations from the model output files and send the observation values back to the master.

As many slave instances of BeoPEST can be started as necessary. Of course, care needs to be taken to avoid conflicts between the slave programs. In particular, the slaves should be run so that different slaves do not attempt to create conflicting files. This is readily achieved by running each slave in a separate directory. Efficient ways of doing this is discussed below.

Running BeoPEST/MPI

BeoPEST/MPI uses the Message Passing Interface (MPI) facilities on the cluster to run. BeoPEST/MPI uses the node numbering (rank) assigned by MPI, with 0 being the master and the remainder being slaves. BeoPEST/MPI supports fault tolerance, heterogeneity and grid computing only to the extent supported by the specific MPI implementation.

BeoPEST/MPI is run using the **mpirun** program provided by the MPI implementation as

mpirun -np N ppest foo /M dir

where **N** is the number of processors to use, **foo** refers to the **foo.pst** pest control file and **dir** is the name of a directory where the model files will be created. BeoPEST/MPI will append the node number to **dir** in order to create a unique file name for each slave. If, for example, **dir** is /tmp/bp, then node 27 will use /tmp/bp27 as its working directory. If the directory does not exist, BeoPEST/MPI will attempt to create the directory. BeoPEST/MPI will set its current

working directory to this directory, so that files with no path is created in this unique directory. Since BeoPEST/MPI will change the directory in which **ppest** is run, it is important that **foo** contains a full path name so that the file can be found after the directory change.

BeoPEST file management

Figure 1 shows the communications between the master and slave instances of BeoPEST, global file storage on the LAN and local file storage on each individual slave. In order to achieve high efficiencies, it is important to organize where files are stored to minimize communications delays in passing messages and reading or writing files, while at the same time avoiding duplicating files with the associated concerns of maintaining up to date copies of each duplicated file.

Running BeoPEST involves four types of files: system specific programs, system specific data files, non system specific data files, and local files.

System specific programs refers to programs such as **ppest** and other PEST utilities. These programs are typically Fortan programs that must be compiled for the specific architecture and operating system. These programs must be accessible on the master and all the slaves, and each slave must be able to access the programs for the appropriate architecture. These programs may be copied to each computer, but more commonly one copy of all the programs can be stored on the local area network and accessed for by all the computers of the same architecture.

Local files are those model input files created by BeoPEST and model output files. These files are by necessity specific to each model run. In order to avoid unnecessary network traffic, these files should be written to scratch space on each slave node. On Unix/Linux machines, for example, this could be in a subdirectory to the /tmp directory. The key here is that for efficiency purposes, these files should be written on locally attached storage on the slave. This is particularly important when the slaves are not on the same local area network as the master. These files will be overwritten many times, and will generally not be retained after the parameter estimation is complete.

The remaining files are input files to PEST and the model. The template and instruction files are required by BeoPEST slaves to know how to process model files. The model may have additional input files that it reads. These files would often be system independent, but in some cases such as when using binary input files, these files may also be system specific. The common thread here is that these files are the same for all model runs. Therefore these

files can be treated as if they are read only. While a copy of these files may be provided on each slave node, it is generally only required that a single copy of these files reside on the local area network. All slaves on the local area network can access these same files. Since these files are only read, most modern operating systems will cache these files so that these files are shared very efficiently on the local area network. Of course, when more than one system type occurs on the local area network, it may be necessary to provide copies of system specific data files for each type of system in such a way that the files can be found in an appropriate directory.

When some of the PEST slaves is at a remote site with a relatively slow connection, it is very important that a local copy of these files are provided that can be efficiently read. For example, when the parameter estimation is done by combining two or more distributed clusters, a local copy of these files should be available on each cluster in order for efficiency. Figure 2 illustrates a BeoPEST run using two remote clusters. Note that each slave communicates directly with the master, but due to the low bandwidth requirements and high latency tolerance of the BeoPEST protocol, this is insignificant. Each slave shown in Figure 2 accesses common files from a local file store on the local area network. Remote copies of these files should be synchronized before the parameter estimation is started, but does not change during the parameter estimation.

When creating shared files such as, for example, the PST file, care should be taken to set up appropriate path names so that the master as well as all the slaves can find all the necessary files. While this can be achieved by different PST files for the master and different groups of slaves, experience has shown that it is generally desirable to set up the different slaves in such a way that a single PST file can be used by all. Mounting a shared disk in the same way on the master and all the slaves typically allows this to be done easily.

Having many copies of input files is generally not good practice as it is difficult to ensure that when a change is made to the input files that the changed file is copied to each instance of the same file on each system. When two or more clusters are used, this is still relatively easy to manage using a command such as **rsync** to synchronize files on each cluster with a master copy. However, to the greatest extent possible, the model runs should be set up to access the same copy of each input file that is not changed as part of the parameter estimation process. This is readily achieved using shared file systems on each cluster and judicious use of path names to allow the master and all the slaves to find the appropriate files. This in particular applies to the files used by BeoPEST slaves such as the instruction and template files, which will always be system independent text files, but is equally true for model

input files that are basically read-only to the model.

Starting Slaves for BeoPEST/TCP

When using BeoPEST/TCP, slaves can be started any time after the master has started. The master will wait indefinitely for slaves to connect if model runs remain. Individual slaves can be started by logging into slave nodes and manually starting the **ppest** program in slave mode. On most clusters, however, the process will be automated.

The **runpest** Perl script can be used to start the master and multiple slaves on a Linux cluster. This script will use the cluster login authentication protocol that allows password-less logins to all nodes in the cluster to start slave programs on each slave node.

Alternately, the master can be started manually and a program such as **mpirun** used to start multiple copies of the slave version. Where multiple clusters are tied together, programs such as **mpirun** can be used on each cluster, but all referencing the same master.

Note that the port number identifies a specific master. It is possible to have multiple masters running simultaneously on the same host, each with its own port. Slaves would then connect to the appropriate master by selecting the desired port.

Security for BeoPEST

When BeoPEST/TCP slaves are run at a remote site, it may be necessary for the slave to tunnel through firewalls at both the remote and local sites. Typically firewalls are set such that outgoing TCP/IP connections are not restricted, but this should be verified. Firewalls typically block incoming connections on unknown ports. Therefore, if port 4004 is used by the master, it will typically be necessary to allow a connection on port 4004 through the firewall.

The values communicated between the slaves and the master are treated as binary numeric values. The bits transmitted are therefore always turned into numbers. A malicious agent could alter these values, which would be catastrophic for the parameter estimation. However, no alteration to these values could cause programs to be run on either the master or slave nodes. The BeoPEST master and slave independently determine how many bytes will be sent and received and will always send or receive only so many bytes. Therefore there is no risk to sending these values across an insecure link.

User authentication is handled outside of BeoPEST. Since the **ppest** program is started manually or using a program such as **mpirun**, user authentication is handled at that level.

The BeoPEST/TCP master program will accept connections from any connecting slave, but since communications with slaves follow a very strict and limited pattern, the security risk is low. The worst risk is a denial of service attack, where a malicious agent pretends to be a client sufficiently closely to delay the master.

A secure connection can always be obtained using the ssh tunneling capabilities.

Scalability issues

BeoPEST was designed to scale well to a large number of clients. With traditional Parallel PEST, the master process could quickly bog down creating input files and reading observations from many clients. BeoPEST avoids this by moving this function to the slaves. This also makes it possible to run slaves at remote sites, because it is no longer necessary for the master to be able to write to the file system where the slave resides. Parameter and observation values are transferred between the master and slave as binary values, which requires very little bandwidth. However, when the master and slave are different architectures, it may be necessary to swap bytes from little endian to big endian or vice versa. This burden is also shifted to the slaves by having slaves adopt the byte gender of the master for all communications.

As a result, as much of the burden as possible for performing the individual model runs are shifted to the slaves. The master only has to perform the parameter estimation calculations. At this time, the size of the matrices being inverted are such that a single master process is not a bottleneck. However, as the number of parameters grow, it may at some point become necessary to also parallelize the matrix calculations involved in the parameter estimation. For a very large number of slaves, the Jacobian calculation may become sufficiently quick that this becomes important, but at this time the Jacobian calculation so dominates the calculation that parallelizing the master calculations becomes an issue. BeoPEST scales well enough to handle very large numbers of slaves, but does not implement parallel processing of the matrix calculations used in the parameter estimation.

The number of slaves to run per physical processor depends on the nature of the problem being solved. Most model runs are CPU bound, so that one slave per physical processor is typically the best way to run BeoPEST. Multi-core processors may support one slave per core. Models that are Input/Output bound may perform better with multiple slaves per processor so that useful computation can be done while waiting for input or output. What combination is optimal would depend on the model at hand, but the one slave per physical processor (or core) is often a good solution.

While the Jacobian is being calculated, the master has little to do. Exchanging parameters and observations with the slaves requires very little effort from the master. Therefore, in an environment with few processors, a slave process can actually be run on the some processor as the master. This will lead to more efficient use of this processor.

When running BeoPEST, the last run with the best parameter can be done on any of the slaves. This may be inconvenient in that the user may want to view the model output. The last model run with the best parameters is by necessity a sequential operation. The **/L** switch on BeoPEST can be used to always have the master make this last model run itself. This will place this last model run in the same directory as all the other PEST output files.

Load Balancing

In order to obtain the full benefit of parallel processing, it is necessary to keep as many of the processors busy doing useful work at all times. Unfortunately, model independent parameter estimation by definition means that the finest level of granularity is that of a model run. In addition, the number of model runs than must be made in order to evaluate the Jacobian is also a function of the number of parameters being estimated.

It can be readily shown that the worst possible situation is when you have *n* equally fast processors and there are *n+1* runs to be made, since while one processor is working on the $n+1^{st}$ run, the other *n-1* processors are idle. This would suggest that adding one more processor will greatly increase the efficiency. However, if this is done by adding a really slow processor, the situation is not helped at all. If, for example, the last processor takes three times as long to run the model as the first *n* processors, the first *n* processors will complete their runs and then sit idle for twice as long while the last procesor finishes its run. It would have been more efficient to have one of those processors complete the $n+1^{st}$ run.

In many cases, the number of model runs are much greater than the number of available processors. In such a case, adding a relatively slow processor may improve the overall computational effort because it is able to make a small contribution to the overall amount of computation that needs to be made. However, if there is one processor that is much slower than the others, there is a very high probability that the faster processors will exhaust the jobs to be run and then have to wait while the slow processor completes its last assigned run. Therefore, it generally doesn't help to add a really slow processor to the slave pool.

A saving grace is that in general model runs are idempotent in the sense that performing the same run more than once is not a problem. Therefore BeoPEST could potentially assign the

last few runs to more than one slaves, and once one of the slaves returns the result, other slaves can be told to abort their runs. This is currently experimental in BeoPEST. Please contact the author for details if this is an desirable feature in your work.

BeoPEST example

Consider the following simple example. A one dimensional flow model named **t1d** is run with input read from **beotest.in**, a set of hydraulic conductivities read from **beotest.dat**, and heads output to **beotest.out**. PEST is used to estimate the hydraulic conductivities based on a set of hydraulic conductivities. The pest control file **beopest.pst** is shown in Figure 3. The file is unremarkable except for the model sections. The master is run in the directory /**pm**/**pest**/**t1d.** The /**pm** file system is mounted on all machines on the local area network.

The model command line reads

```
/pm/pest/t1d/t1d /pm/pest/t1d/beotest.in beotest.dat beotest.out
```

Note that the location of the program **t1d** and the location of the input file **beotest.in** is prefixed by the full path name, so that this command can be executed from any directory on the system, and the executable and data file will still be found. The hydraulic conductivity file **beotest.dat** and model output **beotest.out**, however, is assumed to be in the current working directory.

The model input and output section reads

```
/pm/pest/t1d/beotest.tpl beotest.dat
/pm/pest/t1d/beotest.ins beotest.out
```

Note that the template file **beotest.tpl** and instruction file **beotest.ins** are also in the directory where the **t1d** program, **beotest.in** and **beotest.pst** files reside.

To perform the parameter estimation in sequential mode, run **pest** in the /**pm**/**pest**/**t1d** directory

pest beotest

PEST will perform all the model runs in the local directory, and find those hydraulic conductivities that result in the heads specified.

To perform the parameter estimation using BeoPEST/MPI using a master and 4 slaves, the master and slaves are started by running the **mpirun** program from the /**pm**/**pest**/**t1d** directory as

mpirun -np 5 ppest /pm/pest/t1d/beotest /M /tmp/beotest

The directory /**tmp**/**beotestX** will be created for each slave process where X is the rank

assigned by MPI. This insures that the files are created in local storage and minimize the network traffic generated. The master (rank 0) will run in /**pm**/**pest**/**t1d** and all the pest output files will be saved to that directory. Note that the pest control file is specified using a full path name so that the pest control file can be found after changing directories.

To perform the parameter estimation using BeoPEST/TCP, the instance of BeoPEST that will act as the master is started in the /**pm**/**pest**/**t1d** directory on host **sherkhan** as

ppest beotest /H :4004

Port 4004 is selected as from one of the many unused ports on **sherkhan**. The program will start and within a second or so pause with a message *RUNNING MODEL FOR FIRST TIME .....* This indicates that PEST needs to run the model.

Clients must now be started to run the model. This can be done one the host **alpha** by creating a directory /**tmp**/**1** and making it the current directory. A slave is then started in /**tmp**/**1** as

ppest /pm/pest/t1d/beotest /H sherkhan:4004

BeoPEST knows that this is a slave instance because **sherkhan:4004** is used to specify the name of the master host as well as the port. Note that the name of the PEST control file is given with a full pathname because the slave is run in the directory /**tmp**/**1**. Since the model executable, model data file, template and instruction files in that PEST control file are all specified using full path names, this instance of PEST is able to locate these files as well. However, the hydraulic conductivity file **beopest.dat** created by PEST and the model output file **beopest.out** created by the **t1d** model will reside in /**tmp**/**1**. Once this slave is started, the master will instruct this slave to perform the model runs. The /**tmp** directory on Unix/Linux machines is typically fast local storage and is a convenient place to put temporary run files.

Since host **alpha** actually has two processors, a second instance of BeoPEST can be started. In order to avoid conflicts between the **beopest.dat** and **beopest.out** files, the second slave is started in the directory /**tmp**/**2**. The command used to start BeoPEST is identical

ppest /pm/pest/t1d/beotest /H sherkhan:4004

Although the master now sees two clients on host **alpha**, they are in fact distinct and the master will ask both instances to perform model runs as needed. In fact, if **alpha** had two quad-core processors, it may be able to run eight instances as long as each instance is in a separate directory.

More slave hosts can also be used. Since /**tmp** is local storage, the same naming scheme can be used on hosts **beta**, **gamma**, **delta**, etc. As each instance of BeoPEST is started, it

contacts the master and the master will call upon available slaves to perform the model runs. The master will schedule as many runs as there are available processors, up the maximum number of runs required to calculate the Jacobian. For the lambda search, the master will perform partial parallelization of the lambda search using those processors that are faster than 80% of the fastest processor.

Since BeoPEST uses a parallel lambda search, the result may be slightly different than the result obtained using the sequential lambda search, but in practice these differences are insignificant.

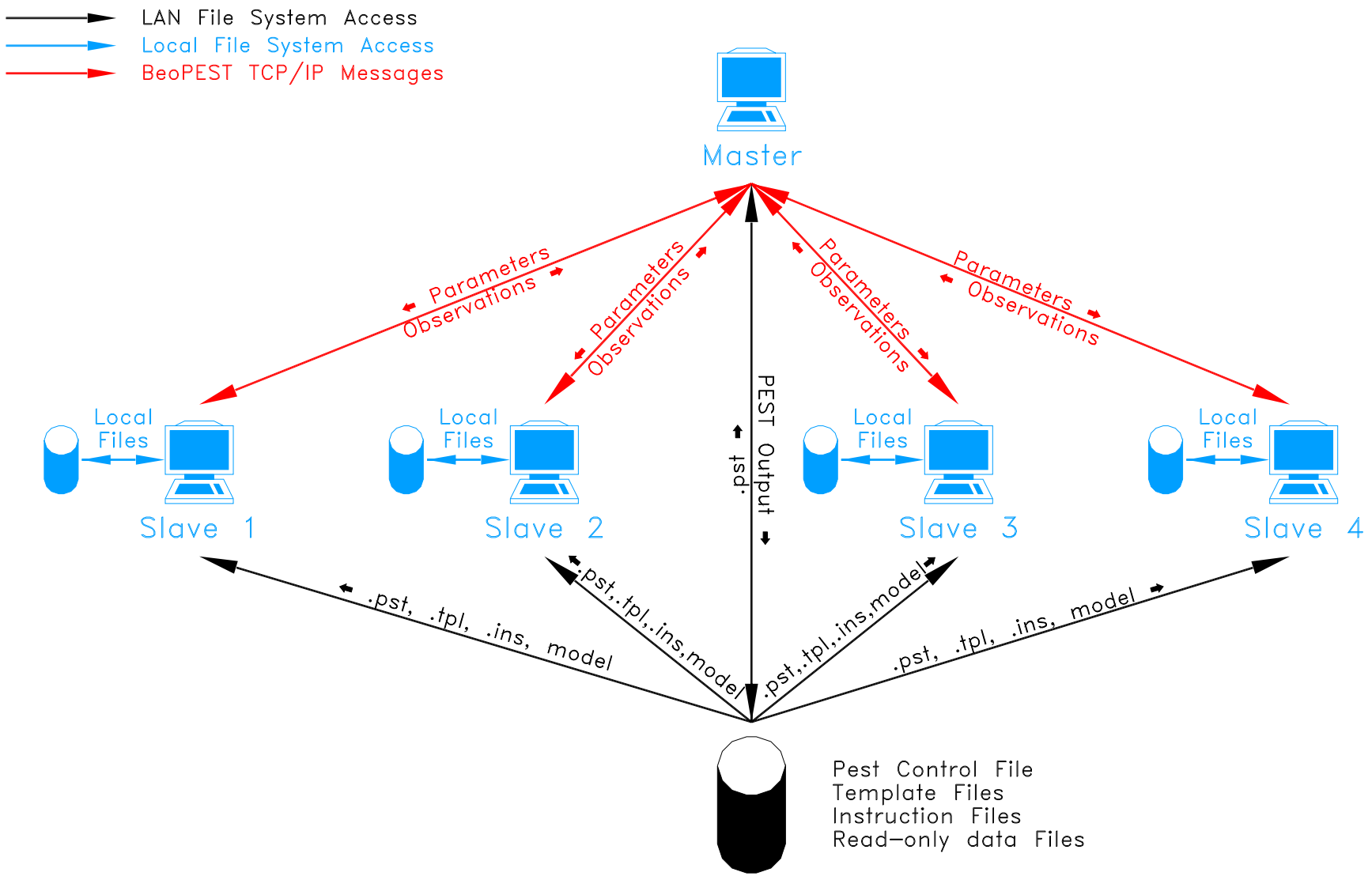

Figure 1. Running BeoPEST on a homogeneous LAN.

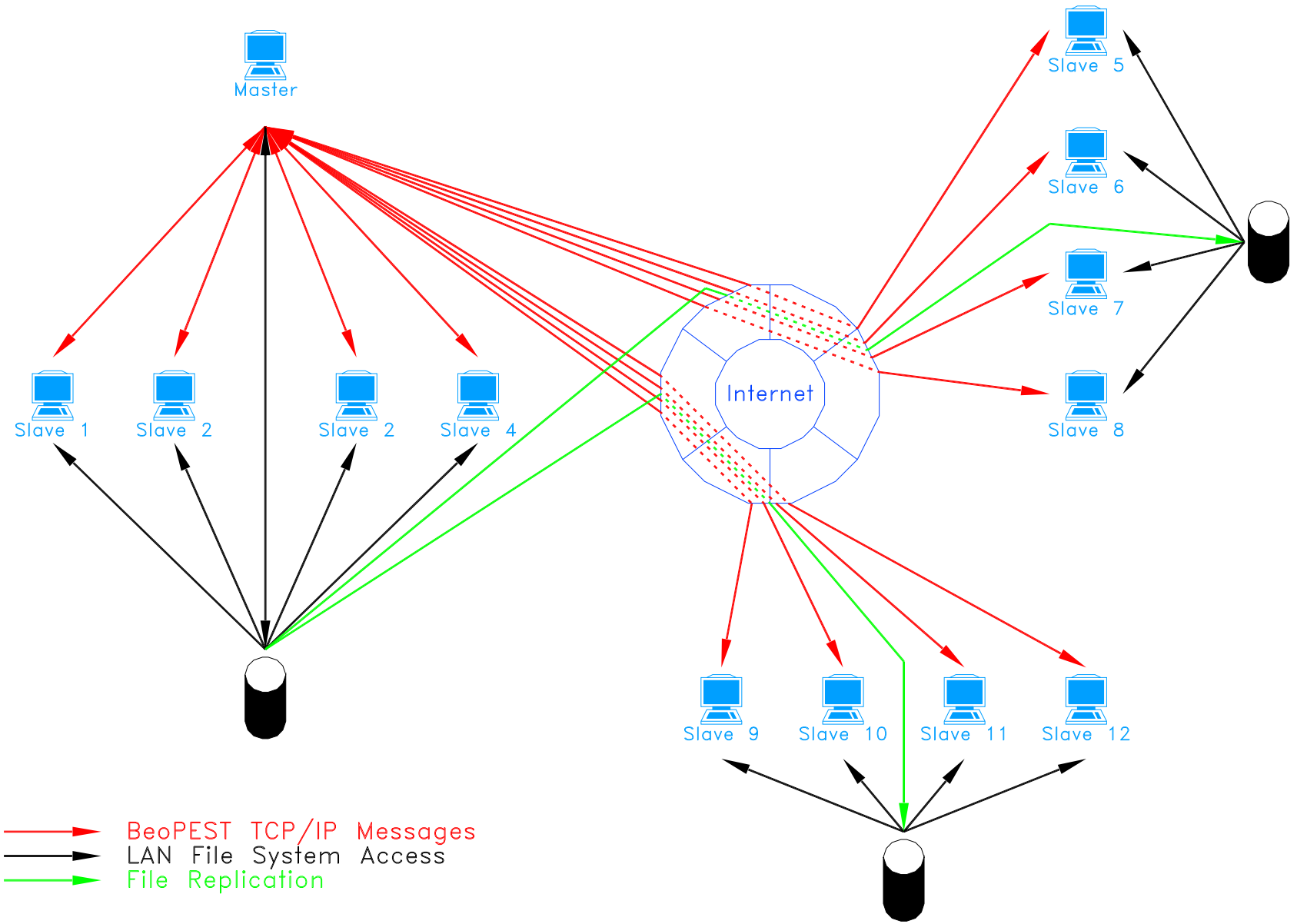

Figure 2. Running BeoPEST across the Internet.

```
pcf
* control data
restart  estimation
   10    10     1     0     1
    1     1 double point   1   0   0
  5.0   2.0   0.3  0.03    10
  3.0   3.0 0.001
  0.1
   30  0.01     3     3  0.01     3
    1     1     1
* parameter groups
K relative 0.01 0.0 switch 2.0 parabolic
* parameter data
k0     fixed relative 1 1.0e-3 1.0e+3 K 1 0 1
k1     log   relative 1 1.0e-3 1.0e+3 K 1 0 1
k2     log   relative 1 1.0e-3 1.0e+3 K 1 0 1
k3     log   relative 1 1.0e-3 1.0e+3 K 1 0 1
k4     log   relative 1 1.0e-3 1.0e+3 K 1 0 1
k5     log   relative 1 1.0e-3 1.0e+3 K 1 0 1
k6     log   relative 1 1.0e-3 1.0e+3 K 1 0 1
k7     log   relative 1 1.0e-3 1.0e+3 K 1 0 1
k8     log   relative 1 1.0e-3 1.0e+3 K 1 0 1
k9     fixed relative 1 1.0e-3 1.0e+3 K 1 0 1
* observation groups
H
* observation data
h0     100.000000 1.0 H
h1      95.532507 1.0 H
h2      82.867842 1.0 H
h3      69.871063 1.0 H
h4      66.179196 1.0 H
h5      62.176584 1.0 H
h6      56.569212 1.0 H
h7      50.553210 1.0 H
h8      41.861036 1.0 H
h9      20.000000 1.0 H
* model command line
/pm/pest/t1d/t1d /pm/pest/t1d/beotest.in beotest.dat beotest.out
* model input/output
/pm/pest/t1d/beotest.tpl beotest.dat
/pm/pest/t1d/beotest.ins beotest.out
* prior information
```

Figure 3. Pest Control File.