High-Throughput Computing Versus High-Performance Computing for Groundwater Applications

Article *in* Ground Water · January 2015

DOI: 10.1111/gwat.12320

2 authors:

Michael N Fienen
United States Geological Survey
124 PUBLICATIONS **3,794** CITATIONS
SEE PROFILE

Randall J Hunt
United States Geological Survey
218 PUBLICATIONS **7,552** CITATIONS
SEE PROFILE

Groundwater

Technical Commentary/

High-Throughput Computing Versus High-Performance Computing for Groundwater Applications

by Michael N. Fienen[1] and Randall J. Hunt[2]

Introduction

High-throughput computing (HTC) is the deployment of resources to tackle a large computational burden where the individual computations do not need to interact while running (Beck 1997). HTC differs from high-performance computing (HPC), where rapid interaction of intermediate results is required to perform the computations. Parallel computing is often equated with HPC and supercomputers, associated with high costs and inaccessibility for those not working at national laboratories or other large institutions. As HPC supercomputing evolved to link many processors in a single facility in 1984, researchers at the University of Wisconsin—Madison took a different approach. They sought to take advantage of idle central processing unit (CPU) cycles (often called "cycle scavenging") on desktop machines spread throughout a single campus department or company or beyond (Thain et al. 2005). The tool they developed was originally called Condor, now called HTCondor (HTCondor Team 2014, http://research.cs.wisc.edu/htcondor/). HTCondor is widely employed in the theoretical physics realm including being used to confirm discovery of the Higgs boson (Ahronovitz 2013). The software is designed to be flexible and general and thus can be used for many other applications.

HTC is best suited for problems that are embarrassingly (also called "pleasingly" parallel; Livny 2011)—those that can be divided among CPUs without the need for communication of intermediate results between CPUs while they run. Beowulf clusters (Becker et al. 1995) build on fast networking of inexpensive individual computers to harness similar power to supercomputers with lower cost. Beowulf clusters have been used in both HPC and HTC contexts. HTCondor was developed to coordinate large numbers of independent parallel runs in an HTC environment. HTC is well suited to any number of available processors, either dedicated or opportunistic, within a company or department. Even cloud computing resources can be efficiently leveraged through HTC and tools such as HTCondor can provide an important link between groundwater practitioners and the power of parallel computing.

Groundwater Applications

In groundwater modeling, there are two primary HTC opportunities—parameter estimation and uncertainty calculations. In nonlinear regression based parameter estimation, such as implemented by the PEST software suite (Doherty 2010), a Jacobian sensitivity matrix is repeatedly calculated to estimate a new set of potentially optimal parameters. The matrix stores the result of the effects of a small perturbation applied to all model parameters on all observations included in the estimation. Therefore, its size, and the number of forward runs of the model required, is defined by the number of observations and parameters included in the parameter estimation. Highly parameterized models (e.g., Hunt et al. 2007; Doherty and Hunt 2010) can require thousands of forward model runs for each Jacobian calculation. However, these runs are independent: only one parameter is perturbed in a run, and its effect is expressed on the observations. Therefore, these runs are ideally suited for HTC in a pleasingly parallel framework. Parallel processing was enhanced in the PEST software suite through BeoPEST (Schreüder 2009) and PEST++ (Welter et al. 2012). The key enhancement removed the requirement that the network between CPUs be defined beforehand, as was required in parallel PEST. That is, at the start of a run, the master does not need information on where each worker is located on a network;

[1]Corresponding author: USGS, Wisconsin Water Science Center, Middleton, WI 53562; (608) 821 3894; fax (608) 821 3817; mnfienen@usgs.gov

[2]USGS, Wisconsin Water Science Center, Middleton, WI 53562.

Received September 2014, accepted December 2014.

doi: 10.1111/gwat.12320

rather, the worker only needs information on where the master resides (Hunt et al. 2010). Typically the Internet TCP/IP protocol is used, so that each worker is launched knowing the master's IP address. When the worker starts, it uses TCP/IP and the master's IP address to tell the master its location and its availability for jobs. The addition of this flexibility means that individual worker processes can be located anywhere with an Internet connection, and communications to and from the master leverage the widely used TCP/IP communications protocol. However, these codes do not handle worker setup, initialization, termination, and clean-up. For small problems these duties can be handled either manually or with utility software (e.g., Karanovic et al. 2012). For larger problems and resources distributed over a wide area network (WAN), the requirements are ideally suited for HTCondor. Its matchmaking abilities find best-suited available resources (those best able to load and execute the requested model runs), transfer all required model and data files, start/stop the workers, and clean up after the run is finished. In the case of the PEST software suite programs, HTCondor handles these duties while the PEST-related code handles run management. This includes sending the workers specific parameter sets to run and receiving and managing the results of each run.

HTCondor can also take on run management duties. One powerful and widely used uncertainty analysis technique is Monte Carlo (MC; e.g., Zheng and Bennett 2002, 353 to 362). MC is based on a stochastic approach where the model inputs are not a single value but are represented as a probability density function (PDF). PDFs are randomly sampled (a "realization"), the model is then run forward using the sampled parameter values, and results of the run are tabulated. Many runs—commonly thousands or hundreds of thousands of runs—are needed for representative sampling. Because each realization is independent of any other realization, MC analysis is a quintessential HTC application. In an example by Juckem et al. (2014), Latin Hypercube sampling was performed using posterior covariance (Starn and Bagtzoglou 2012) estimated by PEST. Realizations were constructed from sampling ranges of parameter values in a calibrated GFLOW (Haitjema 1995) model. From this sampling, several thousand realizations were constructed and then run through HTCondor. HTCondor not only handled worker creation, population, initialization, termination, and clean-up, it also indexed the result of each forward model run. Therefore, HTCondor was the master that handled run management, and compiled the results of the runs upon completion. Figure 1 shows the probabilistic plume extents for a hypothetical wastewater disposal facility (Juckem et al. 2014). Appropriate visualization of model uncertainty is becoming increasingly recognized as an important model output (e.g., Barnett et al. 2012) and MC presentations are well suited for many decisions. The ability to run, manage, and compile model results using HTCondor makes this type of computationally intensive analysis more readily available to practitioners than in the past.

A final example of a groundwater application is the directed acyclic graph (DAG; Jensen and Nielsen 2001). DAGs are workflows of processes in which input for a process can depend on outcomes from one or more other processes. HTCondor can use discrete HTCondor jobs in a DAG to create potentially complex workflows with built-in tools such as DAGMan. DAGMan was used in bgaPEST (Fienen et al. 2013) to calculate Jacobian matrices with monitoring by the parent process. In the MC context, DAGMan could be used to automate what now remains a manual aspect of the process. In the Juckem et al. (2014) example, batches of about 1000 model runs were submitted to be run by HTCondor. A second utility code was run on the results to evaluate convergence of probability values to determine if enough runs were complete to conclude sufficiently exhaustive exploration of probability space. If convergence was not achieved, another batch was submitted to HTCondor and evaluated again. The logic of this process—including intermediate processing of results—could be wrapped in a DAG, obviating the intermediate evaluation during the process.

Nuts and Bolts

HTCondor runs as a master-worker process. The master is a computer that coordinates the matchmaking described above within a local or WAN. The worker machines each run a small daemon (Linux) or service (Windows) in the background that advertises the worker's capability, where an analogy is the classified advertisement section of a newspaper. This daemon/service also handles interactions with the master, accepts jobs that the master matches with available resources, and removes files used in runs to avoid cluttering hard disks with old and extraneous files. The master acts as a matchmaker (Raman et al. 1999), using sophisticated algorithms to negotiate requests for resources, and when a match is made, assigns an appropriate task to that resource, making it as unavailable to the remaining HTCondor "pool" until the job is finished. This kind of matchmaking allows optimization based on computer power (e.g., processor speed, RAM, and hard drive space available), thus allowing the pool of available resources to be used efficiently. For example, low-power machines are given small jobs while high-power machines are used for more intensive jobs. Both master and workers are configured using local ASCII text files. These files reside in an HTCondor directory on the hard drive, and are read when the daemon or service starts.

As specified in the ASCII configuration files, HTCondor operates in three modes to accomplish coordination of resources: (1) as a submit node, (2) as a central manager, and (3) as a worker. These modes can operate on a single machine or on many. Submit nodes are machines from which HTCondor will accept and interpret requests to launch a job. Typically a small subset of resources in an HTCondor pool are given this capability. For the pool, a single CPU acts as a central manager and matchmaker.

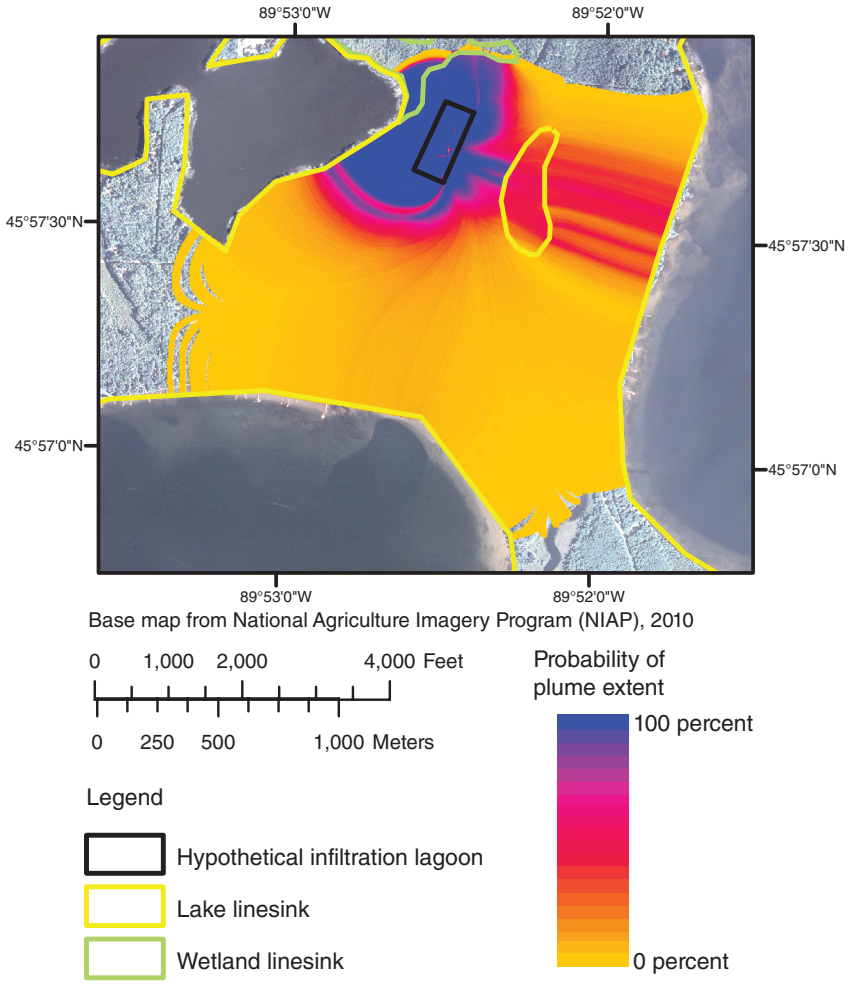

Figure 1. Probabilistic plume extent calculated using Monte Carlo uncertainty analysis on a GFLOW model with HTCondor.

This central manager brokers the requests from submitters and assigns resources from the pool of machines running in the third mode—as a worker. The workers accept input data and executable files, run the requested task, and return results to the submit node.

Within the master-worker framework, a user submits a job request using an ASCII submit file (Figure 2), which is used to inform HTCondor what a user is requesting. This simple ASCII text file identifies how many workers are requested, which data and executable files should be transferred to each worker, what operating system and hardware requirements are needed to perform a job, an executable (typically a shell script [Linux] or batch file [Windows]) to run on each worker, and arguments to pass to each worker for that executable. In the case of PEST, all needed BeoPEST or PEST++ model data files and executables are compressed into a zip file. The submit file tells HTCondor to send out the compressed data file. The batch/shell script file is executed, which unzips the compressed files and starts the BeoPEST/PEST++ worker. In our case the executable is a generic one (Figure 3); the two arguments identify the BeoPEST/PEST++ master IP address and the TCP/IP port to monitor. These are passed to the worker when the

```
notification = Never | in general it's best to avoid email notification when each worker
                       completes its task.
universe = vanilla
log = log/worker_$(Cluster).log | log of all HTCondor jobs-indicates where workers run
output = log/worker_$(Cluster)_$(Process).out | standard out from each worker
error = log/worker_$(Cluster)_$(Process).err | standard error from each worker
executable = worker.bat | indicates the name of the script to run which starts beoPEST
arguments =  127.0.0.1 9999 casename | arguments for worker.bat.
stream_output = True | These two statements indicate that standard error and standard out
stream_error = True  | will both be streamed back to the submit machine
requirements =((Target.OpSys=="WINDOWS")  && (Target.Arch=="X86_64"))
request_memory = 4500 | in this case, if 4.5GB of RAM aren't available, a worker won't start
should_transfer_files = yes | this is vital if runs are performed also on the submit node
transfer_input_files =unzip.exe,data.zip
                       | these input files and worker.bat must be in or refered to from the
                         submit location
queue 77 | the queue
           | indicates the number of workers to start. HTCondor will start as many workers that meet
             the requirements above, up to the number indicated in this variable
```

Figure 2. Example HTCondor submit file workers.sub.

batch file/shell script executes. Once the job is submitted, the files are transferred to each worker much faster than standard operating system copy commands. Thus, hundreds of workers can be populated and started within minutes of a job being submitted.

Standard output and standard error from each worker's console are indexed by worker number and recorded in log files where the master is run. These files are updated in real-time and are useful for debugging failed model runs. HTCondor also allows a global

```
date /T
time /T                  to simplify postauditing of problems, it is good to
dir                      identify time and location of a worker run since
ipconfig                 this will go into the log file

unzip data.zip > nul     uncompress the data file

cd data                  change directory into the data folder

:: %3 is casename, %1 is master IP, %2 is master port
beopest64.exe %3.pst /h %1:%2    call beoPEST with
                                 variables
```

Figure 3. Example script worker.bat called by HTCondor and identified as executable in the submit file.

reporting of the status of the "pool" of resources. This allows one to quickly check the availability of resources, who is running on what resources, and time since each job began. In the first groundwater application example above, both HTCondor and BeoPEST/PEST++ are performing master-worker processes and currently their reporting of the numbering of workers and other information are not coordinated. However, the convenience of submitting an arbitrary number of workers, seeing their output and errors, and having the system clean up after itself far outweigh the sometimes challenging task of determining exact correspondence in bookkeeping between HTCondor and PEST.

Further Opportunities and Possibilities

The workflow discussed here documents a robust and simple application of HTCondor for groundwater problems. Many more opportunities are available and will become available as HTCondor and other codes continue to be improved. HTCondor includes application programming interfaces (APIs) that allow developers to integrate HTCondor run management deeply into programs. Future development of PEST++, for example, could implement these APIs to address the bookkeeping challenges mentioned in the previous section. The possibilities with DAGMan are extensive, including possible optimization of PEST runs such that each Jacobian matrix calculation (very high number of forward runs required) is a job within a DAG monitored by DAGMan. Then unused resources could be released during a PEST parameter upgrade search (usually less than 10 forward runs needed). HTCondor also has excellent job scheduling and priority ranking of submitted jobs. Making use of this functionality involves allowing for eviction of runs when higher priority runs are requested. Full understanding of the ramifications of this must be considered. To handle eviction—at least on Unix-based operating systems—HTCondor supports checkpointing. Checkpointing saves a run in a partially complete state so it can be resumed on another machine. However, incorporating checkpointing into standard Windows-based groundwater programs may not be possible. Geographically separate HTCondor pools can also be linked through "flocking" in which a job submitted in one pool may be run in other pools if allowed. These multiple pools may be various locations within a single organization, or consortia of organizations such as the Open Science Grid (http://www.opensciencegrid.org/). HTCondor was originally developed to scavenge available compute cycles, so it is also possible to configure it to run jobs in the background on idle computers and evict them if the user of that machine moves a mouse or enters a keystroke. Finally HTCondor is becoming "Cloud Computing aware," where the pool of resources can include cloud-based resources. This has the potential to offer virtually unlimited resources—for the price of the rented resources. Over time, HTCondor has evolved and is often (as in our examples) run on dedicated hardware. However, returning to the roots of scavenging cycles has the potential to make every office into a Beowulf cluster, bringing low cost super-computing power to the masses and opening new opportunities for modeling and analysis.

Disclaimer

Any use of trade, product, or firm names is for descriptive purposes only and does not imply endorsement by the U.S. government.

References

Ahronovitz, M. 2013. *HTC, Big data and the God particle* HPCWire, Tabor Communications. http://www.hpcwire.com/2013/03/29/htc_big_data_and_god_particle/ (accessed November 10, 2014).

Barnett, B., L. Townley, V. Post, R. Evans, R.J. Hunt, L. Peeters, S. Richardson, A. Werner, A. Knapton, and A. Boronkay. 2012. Australian groundwater modelling guidelines. Waterlines Report Series No. 82. Canberra, Australia: National Water Commission.

Beck, A. 1997. *High throughput computing: An interview with Miron Livny* HPCWire, Tabor Communications. http://research.cs.wisc.edu/htcondor/HPCwire.1/ (accessed November 10, 2014).

Becker, D.J., T. Sterling, D. Savarese, J.E. Dorband, C.V. Packer, and U.A. Ranawake. 1995. Beowulf: A parallel workstation for scientific computation. In *International Conference on Parallel Processing*, Volume 95.

Doherty, J. 2010. *PEST, Model-independent Parameter Estimation—User Manual*, 5thwith slight additions ed. Brisbane, Australia: Watermark Numerical Computing.

Doherty, J., and R.J. Hunt. 2010. Approaches to highly parameterized inversion: A guide to using pest for groundwater-model calibration. U.S. Geological Survey Scientific Investigations Report 2010-5169, 60 pp. Reston, Virginia: USGS. http://pubs.usgs.gov/sir/2010/5169/

Fienen, M., M. D'Oria, J. Doherty, and R.J. Hunt. 2013. Approaches in highly parameterized inversion: bgaPEST, a Bayesian geostatistical approach implementation with PEST—documentation and instructions. U.S. Geological Survey Techniques and Methods, book 7, section c9, 86 pp. Reston, Virginia: USGS.

Haitjema, H.M. 1995. *Analytic Element Modeling of Groundwater Flow*. San Diego, California: Academic Press.

HTCondor Team. 2014. HTCondor version 8.0 Manual. Technical report, University of Wisconsin-Madison. http://research.cs.wisc.edu/htcondor/ (accessed November 10, 2014).

Hunt, R.J., J. Luchette, W. Schreüder, J. Rumbaugh, J. Doherty, M. Tonkin, and D. Rumbaugh. 2010. Using a cloud to

replenish parched groundwater modeling efforts. *Ground Water* 48, no. 3: 360–365.

Hunt, R.J., J. Doherty, and M. Tonkin. 2007. Are models too simple? Arguments for increased parameterization. *Ground Water* 45, no. 3: 254–261.

Jensen, F.V., and T.D. Nielsen. 2001. *Bayesian Networks and Decision Graphs. Statistics for Engineering and Information Science*. New York: Springer.

Juckem, P.F., M. Fienen, and R.J. Hunt. 2014. Simulation of groundwater flow and interaction of groundwater and surface water on the Lac du Flambeau Reservation, Wisconsin. U.S. Geological Survey Scientific Investigations Report 2014-5020, 34 pp. Reston, Virginia: USGS. http://pubs.usgs.gov/sir/2014/5020/

Karanovic, M., C. Muffels, M. Tonkin, and R.J. Hunt. 2012. Approaches in highly parameterized inversion: PESTCommander, A Graphical User Interface for File and Run Management Across Networks. U.S. Geological Survey Techniques and Methods, book 7, section c, chapter 8, 9 pp. Reston, Virginia: USGS. http://pubs.usgs.gov/tm/tm7c8/

Livny, M. 2011. Personal communication.

Raman, R., M. Livny, and M. Solomon. 1999. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing* 2, no. 2: 129–138.

Schreüder, W. 2009. Running BeoPEST. In *Proceedings from, PEST Conference 2009*, November 1–3, Potomac, MD. Bethesda, Maryland: S.S. Papadopulos and Associates.

Starn, J.J., and A.C. Bagtzoglou. 2012. Programs for calibration-based Monte Carlo simulation of recharge areas. *Ground Water* 50, no. 3: 472–476.

Thain, D., T. Tannenbaum, and M. Livny. 2005. Distributed computing in practice: The Condor experience. *Concurrency and Computation-Practice & Experience* 17, no. 2–4: 323–356.

Welter, D.E., J. Doherty, R.J. Hunt, C. Muffels, M. Tonkin, and W. Schreüder. 2012. Approaches in highly parameterized inversion—PEST++, a Parameter ESTimation code optimized for large environmental models. U.S. Geological Survey Techniques and Methods, book 7, section c5, 47 pp. Reston, Virginia: USGS. http://pubs.usgs.gov/tm/tm7c5/

Zheng, C., and G. Bennett. 2002. *Applied Contaminant Transport Modeling*, 2nd ed. New York: Wiley-Interscience.