Software and Technology

Clustered Computing for Political Science

Andrew D. Martin
Washington University
admartin@wustl.edu

Kevin A. Croker
Washington University
kac3@cec.wustl.edu

Introduction: Why Clusters?

Up and down the halls of universities throughout the world, computers sit in offices and labs doing nothing. Perhaps these computers are looking for alien life as part of the SETI project when not being used, but more likely, they are idle. Have you ever wondered how you could harness these available cycles for your computation needs? Or, perhaps you are tired of using your workstation for model estimation, grinding your other programs to a halt, and are looking for a way to use other available computing power to solve your problem. Some recent advances in clustered computing make putting to use available resources relatively straight forward. Not only does this ameliorate the need to purchase additional hardware, but it makes it possible to harness available spare cycles already available in nearly every academic building.

But why would a political scientist need these sorts of resources? Simply put, many problems in political methodology are computationally expensive. As models get more and more complicated, the computational power needed to estimate them increases. Many likelihood functions are not easily maximized, and various optimizers require significant time to converge. Other optimization problems rely on simulations, sometimes to evaluate the likelihood function (e.g., the use of the Geweke-Hajivassiliou-Keane probability simulator to compute multivariate Normal integrals), other times to use sophisticated genetic algorithms as the optimization routine itself (Mebane and Sekhon 2004). Markov chain Monte Carlo (MCMC) methods—the workhorse of applied Bayesian inference—also require significant computer time, whether in an interpreter such as R or WinBUGS, or in a compiled language like C or FORTRAN. It is not uncommon for political scientists to fit models that take days, weeks, or even months to estimate on state-of-the-art hardware.

Moreover, fitting a single model does not make for good social science. Indeed, all robustness and sensitivity analysis that goes into applied statistical work in political science requires the researcher to fit models with slightly different covariates, perhaps different priors, and different assumed model structures. Having to fit multiple instances of computationally expensive models becomes practically impossible on a single workstation.

In this article we review two state-of-the-art clustering technologies: openMosix and Xgrid. openMosix is used for clustering Linux workstations, and Xgrid is used for clustering MacOS X machines. We are not aware of (nor would we consider using) clustering systems for Windows systems. We have used openMosix for over two years to run a small cluster of Linux machines that we use for running long simulations in R and C++. We have also installed a beta version of Xgrid on some local computers to test its reliability.

Before we describe how each of these technologies works, it is important to draw a distinction between a cluster and a supercomputer. A supercomputer is one physical unit able to process instructions at speeds many times greater than your average workstation. A cluster, on the other hand, is one *virtual* unit consisting of several workstations connected over some sort of network. Supercomputers have highly accelerated input/output, memory access, and processor speed, and typically have shared memory access. As such, they are useful for problems that can be solved in parallel. Supercomputers have been used in political science; Poole and Rosenthal (1997) estimated their two-dimensional D-NOMINATE model on the CYBER 205 supercomputer at Purdue University in the late 1980s. As discussed in the conclusion, clusters can also be used to attack problems that can be implemented in parallel, often at greatly reduced cost. While we have limited experience with explicitly parallel code, in principle this could be used for a number of problems in political science using the same clustering technology discussed below.

openMosix

When constructing our cluster, we chose openMosix[1] for its transparent operation and ease of deployment. Among the other options available at the time, Beowulf and comercial MOSIX, we felt that openMosix represented the best potential for growth and flexibility. How transparent is openMosix? Working on an openMosix cluster is as simple as working on a regular workstation; the user never has

[1] `http://openmosix.sourceforge.net/`

```
11:37am  up 2 days, 30 min,  9 users,
load average: 2.31, 1.12, 0.87 115
processes: 112 sleeping, 3 running, 0 zombie, 0 stopped
CPU states: 792.6% user,  5.4% system,  0.0% nice,  0.0% idle
Mem:  1030332K av, 658444K used, 371888K free,      0K shrd, 6480K buff Swap:
2040244K av,   3272K used, 2036972K free 189600K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT N# MGS %CPU %MEM   TIME COMMAND
25785 adm       17   0  116M 116M  1352 S     4   1 99.9 11.6   1:04 R.bin
25794 adm       15   0  117M 117M   468 S     2   1 99.9 11.6   1:45 R.bin
25758 adm       16   0  116M 116M  1352 S     2   1 99.7 11.6   1:11 R.bin
25776 adm       15   0  115M 115M   468 S     4   2 99.7 11.5   1:23 R.bin
25731 adm       17   0  115M 115M   468 S     3   2 99.2 11.4   1:11 R.bin
25740 adm       17   0  115M 115M   468 S     3   2 99.2 11.5   1:12 R.bin
25749 adm       18   0  115M 115M   468 R     0   2 95.5 11.5   1:00 R.bin
25767 adm       13   0  115M 115M   468 R     0   2 94.7 11.5   0:55 R.bin
22659 adm        9  -1 29420  11M  2232 S     0   0  3.3  1.1   1:02 X
25888 adm        9   0 26196 9796  6852 D     0   0  2.7  0.9   0:02 openmosixv
25887 adm       12   0   892  892   716 R     0   0  1.3  0.0   0:00 mtop
   22 root       9   0     0    0     0 SW    0   0  0.3  0.0   0:21 memsorter
```

Table 1: Output of the openMosix `mtop` command.

to worry about viewing the particular computer system she is on as part of a special entity. When a user starts a program, openMosix looks at all other workstations running openMosix and asks them: "Hey, how busy are you?" The other machines reply, and based on the replies, the user's program is invisibly relocated over the network to the machine where it will run most efficiently. To the user, it still appears as if the process is running locally, but it is actually not. Migration is dynamic and takes place in real-time. openMosix is therefore constantly tuning the performance of all programs that are running on the cluster without ever needing user intervention. Of course, this has limitations, particularly with jobs that are input/output or memory intensive. In these cases, the programs simply stay on the local machine and other programs that can be efficiently relocated are moved. openMosix shines in CPU intensive tasks, which are typical problems in political science.

openMosix is nothing more than an extension to a Linux kernel and is freely available as a fully open-source project. openMosix can be run on a variety of Linux platforms, but for a particular cluster, it is necessary to have homogeneous architectures on all computers in the cluster. (This does not mean that all machines have to be the same; one just cannot mix machines running Linux on PowerPC chips and Pentium chips.) Installation can be as simple as installing the appropriate packages for your Linux distribution (a bare minimum is the openMosix enabled kernel and the userland tools). Once installed, it is easy to see the advantages of openMosix. In Table 1 we show typical output from the `mtop` command, which is a part of the userland tools and is a replacement for the `top` command. Here you will see that we started eight computationally expensive R jobs, which have automatically been migrated to available nodes on our cluster. Each job is pulling nearly 100% of a processor; the N# column shows which of our dual-processor machines the job has migrated to.

openMosix can be used to share resources across a number of workstations. To the extent that most political science departments are not full of Linux users, we suspect most people will use openMosix for clustering dedicated machines. There are also ways to harness spare Windows machines by booting them into Linux (perhaps for a short while) from either a CD or over a network. As an example, our cluster consists of a master node with a number of slave computers which boot off of a private network. Each slave node consists of just a motherboard, power supply, memory, and a network interface card. We have released a set of scripts to build an auto-assimilating cluster (Croker and Martin 2004). The ClusterKnoppix [2] project also has released a number of innovative ways to automatically configure a cluster using network and CD booting. We view the upside of openMosix to be its kernel-level implementation. With it, working on a cluster appears just the same as working on a machine with a large number of processors. Installation and administration are required, but no more so than administering a Linux workstation.

[2] `http://bofh.be/clusterknoppix/`

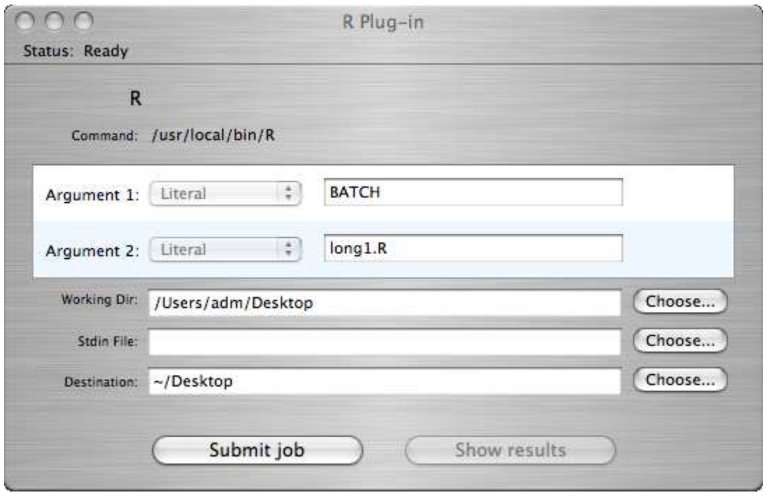
R Plug-in
Status: Ready
R
Command: /usr/local/bin/R
Argument 1: Literal BATCH
Argument 2: Literal long1.R
Working Dir: /Users/adm/Desktop
Stdin File:
Destination: ~/Desktop
Choose...
Submit job
Show results

Figure 1: Xgrid Interface for Submitting an R Program

Xgrid

The Advanced Computing Group at Apple Computer has recently released a package to enable clustering of MacOS X machines.Their technology is called Xgrid [3]. It can be used to harness spare cycles on G4s and G5s that are configured to be part of the cluster. Xgrid is not a kernel-level clustering solution; it resides at the user level, and is a controller/client protocol. Users submit jobs to the controller machine, which distributes them to available client processors. Some clients may be dedicated severs; others might be workstations on the network that have registered with the controller. Xgrid moves jobs automatically, and balances the load across available machines. Each user controls how much, if any, of their processing power they wish to contribute to the cluster. The default setup is to only use workstations when they are not in active use. When a job is completed, the output is returned to the user who submitted the job.

Since Xgrid is not a kernel extension, its installation is as easy as installing a software package. One uses the System Preferences panel to configure the cluster, including setting what machine is the controller, setting up secure passwords to access the controller, the clients, etc. Once configured, the user submits a job to the cluster using a graphical interface. In Figure 1 we show how one would submit a long R job to the cluster. As with openMosix, Xgrid can be used to execute truly parallel programs. As a demonstration, Apple distributes a parallel program used to search DNA and protein databases. The Xgrid program contains a number of graphical tools one can use to see how much available processing power is in use at a given time. The upside to Xgrid for applied work in political science is the availability of hardware; surely more political scientists use MacOS X than Linux. The installation and configuration is trivial, and one can even install Xgrid on ones' colleagues' machines without ever sitting at the terminal (if you have root access). The down side, compared with openMosix, is the batch submission process, which adds an additional step to executing a large program.

Conclusion

In our experience, the time invested in learning and deploying these technologies is well worth it. Being able to estimate ten or more instances of a single model that takes two or three weeks with different priors and covariates *at the same time* considerably speeds up the research process. As our models become increasingly complicated, having technologies that can make computing as easy and quick as possible is of great utility.

Moreover, there are additional gains to be had when our software can take into account these available computation resources by "going parallel." Both openMosix and Xgrid work seemlessly with MPI [4], the protocol most often used to implement parallel algorithms in FORTRAN and C. Due to communication latency, clusters are not quite as fast as traditional supercomputers with shared memory, however they are far more economical. As such, clusters serve as a viable platform for implementing parallel algorithms. We look forward to seeing interesting applications in political science in the future. Happy computing, and happy clustering.

[3]At the moment, Xgrid is freely available at: `http://www.apple.com/acg/xgrid/`
[4]`http://www.lam-mpi.org`