xgrid and R: Parallel Distributed Processing Using Heterogeneous Groups of Apple Computers

by Sarah C. Anoke, Yuting Zhao, Rafael Jaeger and Nicholas J. Horton

Abstract The Apple Xgrid system provides access to groups (or grids) of computers that can be used to facilitate parallel processing. We describe the **xgrid** package which facilitates access to this system to undertake independent simulations or other long-running jobs that can be divided into replicate runs within R. Detailed examples are provided to demonstrate the interface, along with results from a simulation study of the performance gains using a variety of grids. Use of the grid for "embarassingly parallel" independent jobs has the potential for major speedups in time to completion. Appendices provide guidance on setting up the workflow, utilizing add-on packages, and constructing grids using existing machines.

Introduction

Many scientific computations can be sped up by dividing them into smaller tasks and distributing the computations to multiple systems for simultaneous processing. Particularly in the case of *embarrassingly parallel* (Wilkinson and Allen, 1999) statistical simulations, where the outcome of any given simulation is independent of others, parallel computing on existing *grids* of computers can dramatically increase computation speed. Rather than waiting for the previous simulation to complete before moving on to the next, a grid *controller* can distribute *tasks* to *agents* (also known as *nodes*) as quickly as they can process them in parallel. As the number of nodes in the grid increases, the total computation time for a given job will generally decrease. Figure 1 provides a conceptual model of this framework.

Several solutions exist to facilitate parallel computation within R. Wegener et al. (2007) developed **GridR**, a condor-based environment for settings where one can connect directly to agents in a grid. The **Rmpi** package (Yu, 2002) is an R wrapper for the popular Message Passing Interface (MPI) protocol and provides extremely low-level control over grid functionality. The **rpvm** package (Li and Rossini, 2001) provides a connection to a Parallel Virtual Machine (Geist et al., 1994). The **snow** package (Rossini et al., 2007) provides a simpler implementation of **Rmpi** and **rpvm**, using a low-level socket functionality. The **multicore** package (Urbanek, 2011) provides several functions to divide work between a single machine's multiple cores. Starting with release 2.14.0, **snow** and **multicore** are available as slightly revised copies within the **parallel** package in base R.

The Apple Xgrid (Apple Inc., 2009) technology is a parallel computing environment. Many Apple Xgrids already exist in academic settings, and are straightforward to set up. As loosely organized clusters, Apple Xgrids provide *graceful degradation*, where agents can easily be added to or removed from the grid without disrupting its operation. Xgrid supports heterogeneous agents (also a plus in many settings, where a single grid might include a lab, classroom, individual computers, as well as more powerful dedicated servers) and provides automated housekeeping and cleanup. The Xgrid Admin program provides a graphical overview of the controller, agents and jobs that are being managed (instructions on downloading and installing this tool can be found in Appendix C).

We created the **xgrid** package (Horton and Anoke, 2012) to provide a simple interface to this distributed computing system. The package facilitates use of an Apple Xgrid for distributed processing of a simulation with many independent repetitions, by simplifying job submission (or *grid stuffing*) and collation of results. It provides a relatively thin but useful layer between R and Apple's '`xgrid`' shell command, where the user constructs input scripts to be run remotely. A similar set of routines, optimized for parallel estimation of JAGS (just another Gibbs sampler) models is available within the **runjags** package (Denwood, 2010). However, with the exception of **runjags**, none of the previously mentioned packages support parallel computation over an Apple Xgrid.

We begin by describing the **xgrid** package interface to the Apple Xgrid, detailing two examples which utilize this setup, summarizing simulation studies that characterize the performance of a variety of tasks on different grid configurations, then close with a summary. We also include a glossary of terms and provide three appendices detailing how to access a grid using R (Appendix A), how to utilize add-on packages within R (Appendix B), and how to construct a grid using existing machines (Appendix C).

Controlling an Xgrid using R

To facilitate use of an Apple Xgrid using R, we created the **xgrid** package, which contains support rou-

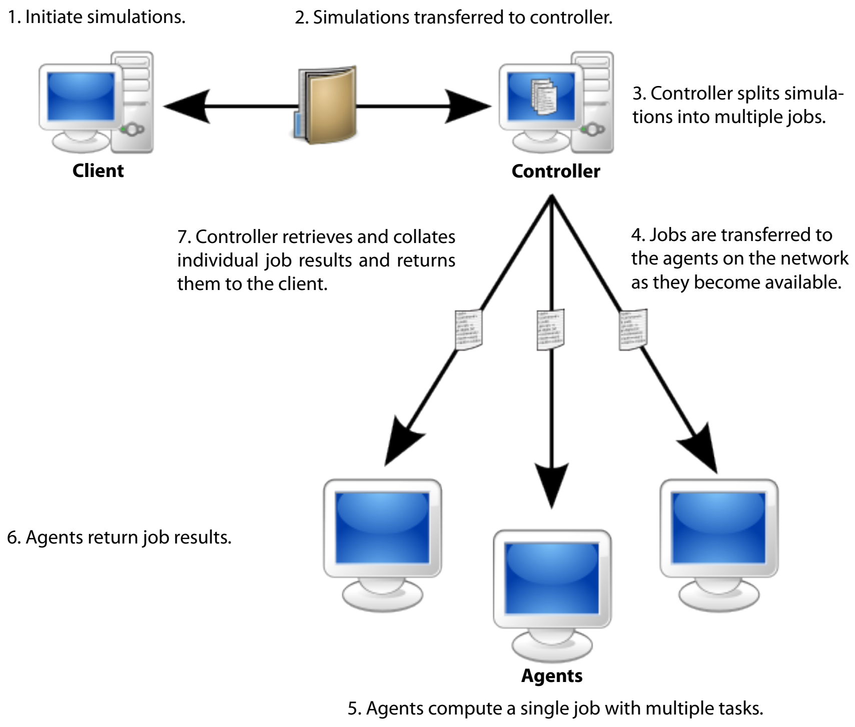

1. Initiate simulations.
2. Simulations transferred to controller.
3. Controller splits simulations into multiple jobs.
Client
Controller
7. Controller retrieves and collates individual job results and returns them to the client.
4. Jobs are transferred to the agents on the network as they become available.
6. Agents return job results.
Agents
5. Agents compute a single job with multiple tasks.

Figure 1: Conceptual model of the Apple Xgrid framework (derived from graphic by F. Loxy)

tines to split up, submit, monitor, then retrieve results from a series of simulation studies.

The `xgrid()` function connects to the grid by repeated calls to the `xgrid` command at the Mac OS X shell level on the client. Table 1 displays some of the actions supported by the `xgrid` command and their analogous routines in the **xgrid** package. While users will ordinarily not need to use these routines, they are helpful in understanding the workflow. These routines are designed to call a specified R script with suitable environment (packages, input files) on a remote machine. The remote job is given arguments as part of a call to '`R CMD BATCH`', which allow it to create a unique location to save results, which are communicated back to the client by way of R object files created with the R `saveRDS()` function. Much of the work involves specifying a naming structure for the jobs, to allow the results to be automatically collated.

The `xgrid()` function is called to start a series of simulations. This function takes as arguments the R script to run on the grid (by default set to 'job.R'), the directory containing input files (by default set to 'input'), the directory to save output created within R on the agent (by default set to 'output'), and a name for the results file (by default set to 'RESULTS.rds'). In addition, the total number of iterations in the simulation (`numsim`) and number of tasks per job (`ntask`) can be specified. The `xgrid()` function divides the total number of iterations into `numsim/ntask` individual jobs, where each job is responsible for calculating the specified number of tasks on a single agent (see Figure 1). For example, if 2,000 iterations are desired, these could be divided into 200 jobs each running 10 of the tasks. The number of active jobs on the grid can be controlled using the `throttle` option (by default, all jobs are submitted then queued until an agent is available). The `throttle` option helps facilitate sharing a large grid between multiple users (since by default the Apple Xgrid system provides no load balancing amongst users).

The `xgrid()` function checks for errors in specification, then begins to repeatedly call the `xgridsubmit()` function for each job that needs to be created. The `xgrid()` function also calls `xgridsubmit()` to create a properly formatted '`xgrid -job submit`' command using Mac OS X through the R `system()` function. This has the effect of executing a command of the form '`R CMD BATCH file.R`' on the grid, with appropriate arguments (the number of repetitions to run, parameters to pass along and the name of the unique filename to save results). The results of the `system()` call are saved to be able to determine the identification number for that particular job. This number can be used to check the status of the job as well as retrieve its results and delete it from the system once it has completed.

Once all of the jobs have been submitted, `xgrid()` then periodically polls the list of active jobs until they are completed. This function makes a call to `xgridattr()` and determines the value of the `jobStatus` attribute. The function waits (sleeps) between each poll, to lessen load on the grid.

When a job has completed, its results are retrieved using `xgridresults()` then deleted from the

Action	R Function	Description
submit	`xgridsubmit()`	submit a job to the grid controller
attributes	`xgridattr()`	check on the status of a job
results	`xgridresults()`	retrieve the results from a completed job
delete	`xgriddelete()`	delete the job

Table 1: Job actions supported by the `xgrid` command and their analogous functions in the **xgrid** package

system using `xgriddelete()`. This capability relies on the properties of the Apple Xgrid, which can be set up to have all files created by the agent when running a given job copied to the 'output' directory on the client computer. When all jobs have completed, the individual result files are combined into a single data frame in the current directory. The 'output' directory has a complete listing of the individual results as well as the R output from the remote agents. This directory can be useful for debugging in case of problems and the contents are typically accessed only in those cases.

To help demonstrate how to access an existing Xgrid, we provide two detailed examples: one involving a relatively straightforward computation assessing the robustness of the one-sample t-test and the second requiring use of add-on packages to undertake simulations of a latent class model. These examples are provided as vignettes within the package. In addition, the example files are available for download from http://www.math.smith.edu/xgrid.

Examples

Example 1: Assessing the robustness of the one-sample t-test

The t-test is remarkedly robust to violations of its underlying assumptions (Sawiloswky and Blair, 1992). However, as Hesterberg (2008) argues, not only is it possible for the total non-coverage to exceed α, the asymmetry of the test statistic causes one tail to account for more than its share of the overall α level. Hesterberg found that sample sizes in the thousands were needed to get symmetric tails.

In this example, we demonstrate how to utilize an Apple Xgrid cluster to investigate the robustness of the one-sample t-test, by looking at how the α level is split between the two tails. When the number of simulation iterations is small ($<$ 100,000), this study runs very quickly as a loop in R. Here, we provide the computation of a study consisting of 10^6 iterations. A more efficient alternative would be to use `pvec()` or `mclapply()` of the **multicore** package to distribute this study over the cores of a single machine. However for the purposes of illustration, we conduct this simple statistical investigation over an Xgrid to demonstrate package usage, and to compare the results and computation time to the same study run with a `for` loop on a local machine.

Our first step is to set up an appropriate directory structure for our simulation (see Figure 5; Appendix A provides an overview of requirements). The first item is the folder 'input', which contains two files that will be run on the remote agents. The first of these files, 'job.R' (Figure 2), defines the code to run a particular task, `ntask` times.

In this example, the `job()` function begins by generating a sample of `param` exponential random variables with mean 1. A one-sample t-test is conducted on this sample and logical (`TRUE/FALSE`) values denoting whether the test rejected in that tail are saved in the vectors `leftreject` and `rightreject`. This process is repeated `ntask` times, after which `job()` returns a data frame with the rejection results and the corresponding sample size.

```
# Assess the robustness of the one-sample
# t-test when underlying data are exponential.
# This function returns a data frame with
# number of rows equal to the value of "ntask".
# The option "param" specifies the sample size.
job <- function(ntask, param) {
   alpha <- 0.05  # how often to reject under null
   leftreject <- logical(ntask)   # placeholder
   rightreject <- logical(ntask)  # for results
   for (i in 1:ntask) {
      dat <- rexp(param)  # generate skewed data
      left <- t.test(dat, mu = 1,
         alternative = "less")
      leftreject[i] <- left$p.value <= alpha/2
      right <- t.test(dat, mu = 1,
         alternative = "greater")
      rightreject[i] <- right$p.value <= alpha/2
   }
   return(data.frame(leftreject, rightreject,
      n = rep(param, ntask)))
}
```

Figure 2: Contents of 'job.R'

The folder 'input' also contains 'runjob.R' (Figure 3), which retrieves command line arguments generated by `xgrid()` and passes them to `job()`. The results from the completed job are saved as `res0`, which is subsequently saved to the 'output' folder.

The folder 'input' may also contain other files (including add-on packages or other files needed for the simulation; see Appendix B for details).

```
source("job.R")
# commandArgs() is expecting three arguments:
# 1) number of tasks to run within this job
# 2) parameter to pass to the function
# 3) place to stash the results when finished
args     <- commandArgs(trailingOnly = TRUE)
ntask1   <- as.numeric(args[1])
param1   <- args[2]
resfile  <- args[3]
res0     <- job(ntask = ntask1, param = param1)
# stash the results
saveRDS(res0, file = resfile)
```

Figure 3: Contents of 'runjob.R'

The next item in the directory structure is 'simulation.R' (Figure 4), which contains R code to be run on the client machine using `source()`. The call to `xgrid()` submits the simulation to the grid for calculation, which includes passing `param` and `ntask` to `job()` within 'job.R'. Results from all jobs are returned as one data frame, `res`. The call to `with()` summarizes all results in a table and prints them to the console.

```
require(xgrid)
# run the simulation
res <- xgrid(Rcmd = "runjob.R", param = 30,
   numsim = 10^6, ntask = 5*10^4)
# analyze the results
with(res, table(leftreject,rightreject))
```

Figure 4: Contents of 'simulation.R'

Here we specify a total of 10^6 simulation iterations, to be split into twenty jobs of 5×10^4 simulations each. Note that the number of jobs is calculated as the total number of simulation iterations (`numsim`) divided by the number of tasks per job (`ntask`). Each simulation iteration has a sample size of `param`.

The final item in the directory structure is 'output'. Initially empty, results returned from the grid are saved here (this directory is automatically created if it does not already exist).

Figure 5 displays the file structure within the directory used to access the Xgrid.

Jobs are submitted to the grid by running 'simulation.R'. In this particular simulation, twenty jobs are submitted. As jobs are completed, the results are saved in the 'output' folder then removed from the grid.

Figures 6 and 7 display management tools available using the Xgrid Admin interface.

In addition to returning a data frame (10^6 rows and 3 columns) with the collated results, the `xgrid()` function saves this object as a file (by default as `res` in the file 'RESULTS.rds').

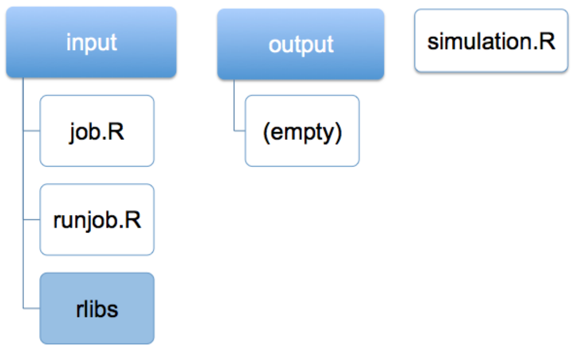
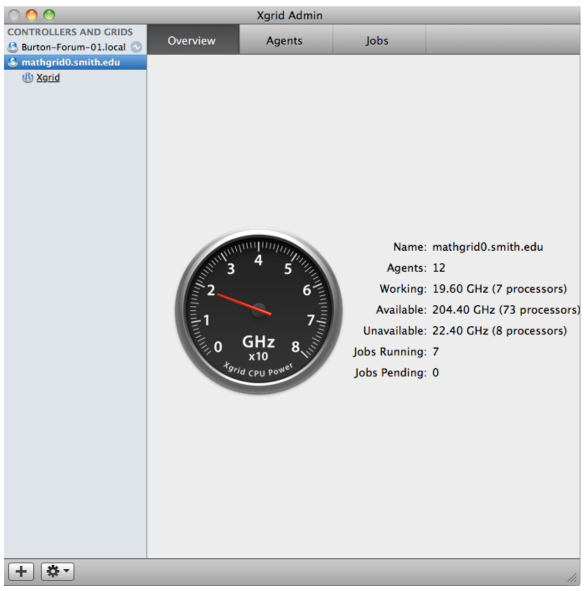
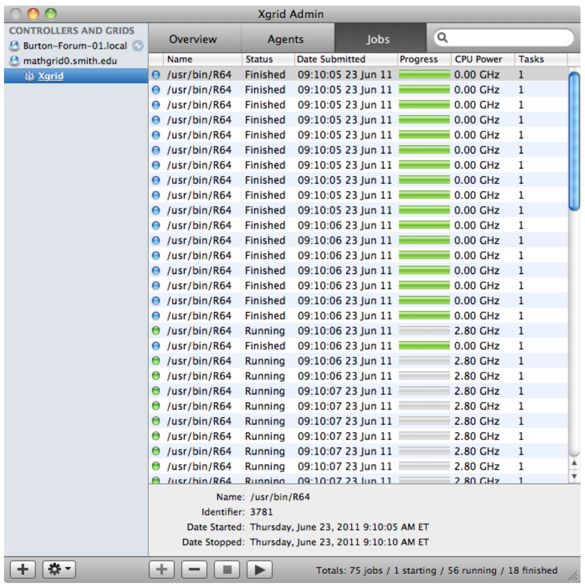


Figure 5: File structure needed on the client to access the Xgrid (the 'rlibs' folder contains any add-on packages required by the remote agent)



Name: mathgrid0.smith.edu
Agents: 12
Working: 19.60 GHz (7 processors)
Available: 204.40 GHz (73 processors)
Unavailable: 22.40 GHz (8 processors)
Jobs Running: 7
Jobs Pending: 0

Figure 6: Monitoring overall grid status using the Xgrid Admin application



Name	Status	Date Submitted	Progress	CPU Power	Tasks
/usr/bin/R64	Finished	09:10:05 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:05 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:05 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:05 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:05 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:05 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:05 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:05 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:05 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:05 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:06 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:06 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:06 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:06 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:06 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:06 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Finished	09:10:06 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Running	09:10:06 23 Jun 11	⬜	2.80 GHz	1
/usr/bin/R64	Finished	09:10:06 23 Jun 11	🟩	0.00 GHz	1
/usr/bin/R64	Running	09:10:06 23 Jun 11	⬜	2.80 GHz	1
/usr/bin/R64	Running	09:10:06 23 Jun 11	⬜	2.80 GHz	1
/usr/bin/R64	Running	09:10:07 23 Jun 11	⬜	2.80 GHz	1
/usr/bin/R64	Running	09:10:07 23 Jun 11	⬜	2.80 GHz	1
/usr/bin/R64	Running	09:10:07 23 Jun 11	⬜	2.80 GHz	1
/usr/bin/R64	Running	09:10:07 23 Jun 11	⬜	2.80 GHz	1
/usr/bin/R64	Running	09:10:07 23 Jun 11	⬜	2.80 GHz	1
/usr/bin/R64	Running	09:10:07 23 Jun 11	⬜	2.80 GHz	1
/usr/bin/R64	Running	09:10:07 23 Jun 11	⬜	2.80 GHz	1

Name: /usr/bin/R64
Identifier: 3781
Date Started: Thursday, June 23, 2011 9:10:05 AM ET
Date Stopped: Thursday, June 23, 2011 9:10:10 AM ET

Totals: 75 jobs / 1 starting / 56 running / 18 finished

Figure 7: Job management using Xgrid Admin

		Class				
Criteria	Acronym	2	3	**4**	5	6+
Bayesian Information Criterion	BIC	49%	44%	**7%**	0%	0%
Akaike Information Criterion	AIC	0%	0%	**53%**	31%	16%
Consistent Akaike Information Criterion	cAIC	73%	25%	**2%**	0%	0%
Sample Size Adjusted Bayesian Information Criterion	aBIC	0%	5%	**87%**	6%	2%

Table 2: Percentage of times (out of 100 simulations) that a particular number of classes was selected as the best model (where the true data derive from 4 distinct and equiprobable classes, simulation sample size 300), reprinted from Swanson et al. (2011). Note that a perfect criterion would have "100%" under class 4.

In terms of our motivating example, when the underlying data are normally distributed, we would expect to reject the null hypothesis 2.5% of the time on the left and 2.5% on the right. The simulation yielded rejection rates of 6.5% and 0.7% for left and right, respectively. This confirms Hesterberg's argument regarding lack of robustness for both the overall α-level as well as the individual tails.

Regarding computation time, this simulation took 48.2 seconds (standard deviation of 0.7 seconds) when run on a heterogeneous mixture of twenty iMacs and Mac Pros. When run locally on a single quad-core Intel Xeon Mac Pro computer, this simulation took 592 seconds (standard deviation of 0.4 seconds).

Example 2: Fitting latent class models using add-on packages

Our second example is more realistic, as it involves simulations that would ordinarily take days to complete (as opposed to seconds for the one-sample t-test). It involves study of the properties of latent class models, which are used to determine better schemes for classification of eating disorders (Keel et al., 2004). The development of an empirically-created eating disorder classification system is of public health interest as it may help identify individuals who would benefit from diagnosis and treatment.

As described by Collins and Lanza (2009), latent class analysis (LCA) is used to identify subgroups in a population. There are several criteria used to evaluate the fit of a given model, including the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC), the Consistent Akaike Information Criterion (cAIC), and the Sample Size Adjusted Bayesian Information Criterion (aBIC). These criteria are useful, but further guidance is needed for researchers to choose between them, as well as better understand how their accuracy is affected by methodological factors encountered in eating disorder classification research, such as unbalanced class size, sample size, missing data, and under- or over-specification of the model. Swanson et al. (2011) undertook a comprehensive review of these model criteria, including a full simulation study to generate hypothetical data sets, and investigated how each criterion behaved in a variety of statistical environments. For this example, we replicate some of their simulations using an Apple Xgrid to speed up computation time.

Following the approach of Swanson et al., we generated "true models" where we specified an arbitrary four-class structure (with balanced number of observations in each class). This structure was composed of 10 binary indicators in a simulated data set of size 300. The model was fitted using the `poLCA()` function of the **poLCA** (polytomous latent class analysis) package (Linzer and Lewis, 2011). Separate latent class models were fitted specifying the number of classes, ranging from two to six. For each simulation, we determined the lowest values of BIC, AIC, cAIC and aBIC and recorded the class structure associated with that value.

Swanson and colleagues found that for this set of parameter values, the AIC and aBIC picked the correct number of classes more than half the time (see Table 2).

This example illustrates the computational burden of undertaking simulation studies to assess the performance of modern statistical methods, as several minutes are needed to undertake each of the single iterations of the simulation (which may explain why Swanson and colleagues only ran 100 simulations for each of their scenarios).

A complication of this example is that fitting LCA models in R requires the use of add-on packages. For instance, we use **poLCA** and its supporting packages with an Apple Xgrid to conduct our simulation. When a simulation requires add-on packages, they must be preloaded within the job and shipped over to the Xgrid. The specific steps of utilizing add-on packages are provided in Appendix B.

Simulation results

To better understand the potential performance gains when running simulations on the grid, we assessed the relative performance of a variety of grids with different characteristics and differing numbers of tasks per job (since there is some overhead to the

Grid description	ntask	numjobs	mean	sd
Mac Pro (1 processor)	1,000	1	43.98	0.27
Mac Pro (8 processors)	20	50	9.97	0.04
Mac Pro (8 processors)	10	100	9.65	0.05
Mac Pro (8 processors)	4	250	9.84	0.59
grid of 11 machines (66 processors)	1	1,000	1.02	0.003
grid of 11 machines (77 processors)	1	1,000	0.91	0.003
grid of 11 machines (88 processors)	20	50	1.28	0.01
grid of 11 machines (88 processors)	10	100	1.20	0.04
grid of 11 machines (88 processors)	8	125	1.04	0.03
grid of 11 machines (88 processors)	4	250	0.87	0.01
grid of 11 machines (88 processors)	1	1,000	0.87	0.004

Table 3: Mean and standard deviation of elapsed timing results (in hours) to run 1,000 simulations (each with three repetitions) using `poLCA()`, by grid and number of tasks per job. Note that the Mac Pro processors were rated at 3 GHz while the grid CPUs were rated at 2.8 GHz.

Apple Xgrid system). One grid was a single quad-core Intel Xeon Mac Pro computer (two quad-core processors, for total of 8 processors, each rated at 3 GHz, total 24 GHz). The other was a grid of eleven rack-mounted quad-core Intel servers (a total of 88 processors, each rated at 2.8 GHz, total 246.4 GHz). For comparison, we also ran the job directly within R on the Mac Pro computer (equivalent to '`ntask = 1000,numjobs = 1`'), as well as on the 88 processor grid but restricting the number of active jobs to 66 or 77 using the `throttle` option within `xgrid()`. For each setup, we varied the number of tasks (`ntask`) while maintaining 1,000 simulations for each scenario (`ntask * numjobs = 1000`).

For simulations with a relatively small number of tasks per job and a large number of jobs, the time to submit them to the grid controller can be measured in minutes; this overhead may be nontrivial. Also, for simulations on a quiescent grid where the number of jobs is not evenly divisible by the number of available processors, some part of the grid may be idle near the end of the simulation and the elapsed time longer than necessary.

Each simulation was repeated three times and the mean and standard deviation of total elapsed time (in hours) was recorded. Table 3 displays the results.

As expected, there was considerable speedup by using the grid. When moving from a single processor to 8, we were able to speed up our simulation by 34 hours – a 78% decrease in elapsed time [(43.98-9.84)/43.98]. We saw an even larger decrease of 98% when moving to 88 (somewhat slower) processors. There were only modest differences in the elapsed time comparing different configurations of the number of tasks per job and number of jobs, indicating that the overhead of the system imposes a relatively small cost. Once the job granularity is sufficiently fine, performance change is minimal.

Summary

We have described an implementation of a "grid stuffer" that can be used to access an Apple Xgrid from within R. Xgrid is an attractive platform for scientific computing because it can be easily set up, and it handles tedious housekeeping and collation of results from large simulations with relatively minimal effort. Core technologies in Mac OS X and easily available extensions simplify the process of creating a grid. An Xgrid may be configured with little knowledge of advanced networking technologies or disruption of networking activities.

Another attractive feature of this environment is that the Xgrid degrades gracefully. If an individual agent fails, then the controller will automatically resubmit the job to another agent. Once all of the submitted jobs are queued up on a controller, the `xgrid()` function can handle temporary controller failures (if the controller restarts then all pending and running jobs will be restarted automatically). If the client crashes, the entire simulation must be restarted. It may be feasible to keep track of this state to allow more fault tolerance in a future release.

A limitation of the Apple Xgrid system is that it is proprietary, which may limit its use. A second limitation is that it requires some effort on the part of the user to structure their program in a manner that can be divided into chunks then reassembled. Because of the relatively thin communication connections between the client and controller, the user is restricted to passing configuration via command line arguments in R, which must be parsed by the agent. Results are then passed back via the filesystem. In addition to the need for particular files to be created that can be run on the client as well as on the controller, use of the Apple Xgrid system may also require separate installation of additional packages needed for the simulation (particularly if the user does not have administrative access or control of the individual agents).

However, as demonstrated by our simulation studies, the Apple Xgrid system is particularly well-suited for embarrassingly parallel problems (e.g. simulation studies with multiple independent runs). It can be set up on a heterogeneous set of machines (such as a computer classroom) and configured to run when these agents are idle. For such a setup, the overall process load tends to balance if the simulation is broken up into sufficiently fine grained jobs, as faster agents will take on more than their share of jobs.

The Xgrid software is available as an additional download for Apple users as part of Mac OS X Server, and it is straightforward to set up a grid using existing computers (instructions are provided in Appendix C). In our tests, it took approximately 30 minutes to set up an Apple Xgrid on a simple network consisting of three computers.

Our implementation provides for three distinct authentication schemes (Kerberos, clear text password, or none). While the Xgrid system provides full support for authentication using Kerberos, this requires more configuration and is beyond the scope of this paper. The system documentation (Apple Inc., 2009) provides a comprehensive review of administration using this mechanism. If the 'auth = "Password"' option is used, then the XGRID_CONTROLLER_PASSWORD environment variable must be set by the user to specify the password. As with any shared networking resource, care should be taken when less secure approaches are used to access the controller and agents. Firewalling the Xgrid controller and agents from outside networks may be warranted.

There are a number of areas of potential improvement for this interface. It may be feasible to create a single XML file to allow all of the simulations to be included as multiple tasks within a single job. This has the potential to decrease input/output load and simplify monitoring.

As described previously, the use of parallel computing to speed up scientific computation using R by use of multiple cores, tightly connected clusters, and other approaches remains an active area of research. Other approaches exist, though address different issues. For example, the **GridR** package does not support access through the xgrid command and requires access to individual agents. The **runjags** package provides an alternative interface that is particularly well suited for Bayesian estimation using Gibbs sampling.

Our setup focuses on a simple (but common) problem: embarrassingly parallel computations that can be chopped into multiple tasks or jobs. The ability to dramatically speed up such simulations within R by running them on a designated grid (or less formally on a set of idle machines) may be an attractive option in many settings with Apple computers.

Glossary

Agent A member of a grid that performs tasks distributed to it by the controller (also known as a node). An agent can be its own controller.

Apple Remote Desktop (ARD) An application that allows a user to monitor or control networked computers remotely.

Bonjour networking Apple's automatic network service discovery and configuration tool. Built on a variety of commonly used networking protocols, Bonjour allows Mac computers to easily communicate with each other and with a large number of compatible devices, with little to no user configuration required.

Client A computer that submits jobs to the controller for grid processing.

Controller A computer (running OS X Server or XgridLite) that accepts jobs from clients and distributes them to agents. A controller can also be an agent within the grid.

Embarassingly parallel A computation that can be divided into a series of independent tasks where little or no communication is required.

Grid More general than an Apple Xgrid, a group of machines (agents), managed by a controller, which accept and perform computational tasks.

Grid stuffer An application that submit jobs and retrieves their results through the xgrid command. The xgrid() function within the **xgrid** package implements a grid stuffer in R.

Job A group of one or more tasks submitted by a client to the controller.

Mac OS X Server A Unix server operating system from Apple, that provides advanced features for operating an Xgrid controller (for those without Mac OS X Server, the XgridLite panel provides similar functionality for grid computing).

plist file Short for "property list file". On Mac OS computers, this is an XML file that stores user- or system-defined settings for an application or extension. For example, the "Preferences" pane in an application is linked to a plist file such that when Preferences are changed, their values are recorded in the plist file, to be read by other parts of the application when running.

Preference pane Any of the sections under the Mac OS X System Preferences, each controlling a different aspect of the operating system (desktop background, energy saving preferences, etc.) or of user-installed programs. XgridLite is a preference pane, since it is simply a GUI for core system tools (such as the `xgridctl` command).

Processor core Independent processors in the same CPU. Most modern computers have multi-core CPUs, usually with 2 or 4 cores. Multi-core systems can execute more tasks simultaneously (in our case, multiple R runs at the same time). Multi-core systems are often able to manage tasks more efficiently, since less time is spent waiting for a core to be ready to process data.

Task Sub-components of jobs. A job may contain many tasks, each of which can be performed individually, or may depend on the results of other tasks or jobs.

Xgrid Software and distributed computing protocol developed by Apple that allows networked computers to contribute to distributed computations.

XgridLite Preference pane authored by Ed Baskerville. A free GUI to access Xgrid functionality that is built into all Mac OS X computers. Also facilitates configuration of controllers (similar functionality is provided by Mac OS X Server).

Appendix A: Accessing the Apple Xgrid using R

This section details the steps needed to access an existing Xgrid using R. All commands are executed on the client.

Install the package from CRAN

The first step is to install the package from CRAN on the client computer (needs to be done only once)

```
install.packages("xgrid")
```

Determine hostname and access information for the Apple Xgrid

The next step is to determine the access procedures for your particular grid. These include the hostname (specified as the `grid` option) and authorization scheme (specified as the `auth` option).

Create directory and file structure

An overview of the required directory structure is provided in Figure 5.

The 'input' folder contains two files, 'job.R' and 'runjob.R'. The file 'job.R' contains the definition for the function `job()`. This function expects two arguments: `ntask`, which specifies the number of tasks per job, and `param`, used to define any parameter of interest. Note that `job()` must have `ntask` and `param` as arguments, but can be modified to take more. This function returns a data frame, where each row is the result of one task. The file 'runjob.R' (running on the agent) receives three arguments when called by `xgrid()` (running on the client): `ntask`, `param`, and `resfile` (see description of 'simulation.R' below). The arguments `ntask` and `param` are passed to `job()`, and the data frame returned by `job()` is saved to the folder 'output', with filename specified by `resfile`.

The folder 'output' will contain the results from the simulations. If created manually, it should be empty. If not created manually, the function `xgrid()` will create it.

The script 'simulation.R' accesses the controller by calling `xgrid()` and allows the user to override any default arguments.

Call `xgrid()` within 'simulation.R'

After loading the **xgrid** package and calling the `xgrid()` function as described in 'simulation.R', results from all `numsim` simulations are collated and returned as the object `res`. This object is also saved as the file 'RESULTS.rds', at the root of the directory structure. This can be analyzed as needed.

Appendix B: Using additional packages with the Apple Xgrid

One of the strengths of R is the large number of add-on packages that extend the system. If the user of the grid has administrative privileges on the individual machines then any needed packages can be installed in the usual manner.

However, if no administrative access is available, it is possible to utilize such packages within this setup by manually installing them in the 'input/rlibs' directory and loading them within a given job.

This appendix details how to make a package available to a job running on a given agent.

1. Install the appropriate distribution file from CRAN into the directory 'input/rlibs'. This can be done by using the `lib` argument of `install.packages()`. For instance, in Example 2 we can install **poLCA** by using the call `install.packages("poLCA",lib = "~/.../input/rlibs")`, where '~/.../' emphasizes use of the absolute path to the 'rlibs' directory.

2. Access the add-on package within a job running on an agent. This is done by using the `lib.loc` option within the standard invocation of `library()`. For instance, in Example 2 this can be done by using the call `library(poLCA,lib.loc="./rlibs")`.

It should be noted that the package will need to be shipped over to the agent for each job, which may be less efficient than installing the package once per agent in the usual manner (but the latter option may not be available unless the grid user has administrative access to the individual agents).

Appendix C: Xgrid setup

In this section we discuss the steps required to set up a new Xgrid using XgridLite under Mac OS X 10.6 or 10.7. We presume some background in system administration and note that installation of a grid will likely require the assistance of technical staff.

Depending on the nature of the network and machines involved, different Xgrid-related tools can be used, including Mac OS X Server and XgridLite. Mac OS X Server has several advantages, but for the purposes of this paper, we restrict our attention to XgridLite.

Anatomy of an Apple Xgrid

It is important to understand the flow of information through an Apple Xgrid and the relationship between the different computers involved. The centerpoint of an Apple Xgrid is the controller (Figure 1). When clients submit jobs to the grid, the controller distributes them to agents, which make themselves available to the controller on the local network (using Apple's Bonjour network discovery). That is, rather than the controller keeping a master list of which agents are part of the grid, agents detect the presence of a controller on the network and, depending on their configuration, make themselves available for job processing.

This loosely coupled grid structure means that if an agent previously involved in computation is unavailable, the controller simply passes jobs to other agents. This functionality is known as graceful degradation. Once an agent has completed its computation, the results are made available to the controller, where the client can retrieve them. As a result, the client never communicates directly with agents, only with the controller.

Downloading XgridLite

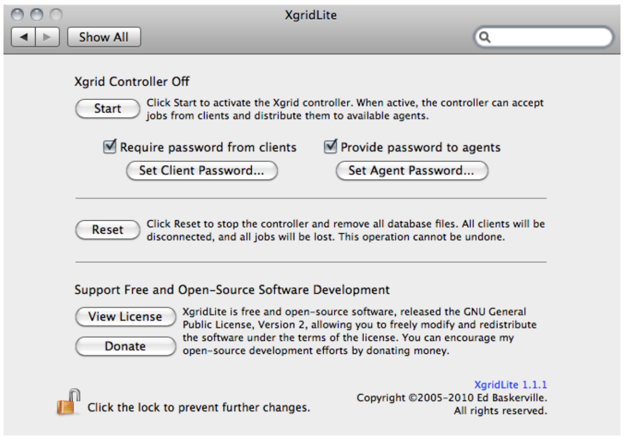
XgridLite (`http://edbaskerville.com/software/xgridlite`) is free and open-source software, released under the GNU GPL (general public license), and installed as a Mac OS X preference pane. Once it is installed, it is found in the "Other" section (at the bottom) of the System Preferences (Apple menu → System Preferences).

Downloading server tools

Although not necessary to install and operate an Apple Xgrid, the free Apple software suite called Mac OS X Server Tools is extremely useful. In particular, the Xgrid Admin application (see Figure 6) allows for the monitoring of any number of Apple Xgrids, including statistics on the grid such as the number of agents and total processing power, as well as a list of jobs which can be paused, stopped, restarted, and deleted. Most "grid stuffers" (our system included) are designed to clean up after themselves, so the job management functions in Xgrid Admin are primarily useful for monitoring the status of jobs.

Setting up a controller

Figure 8 displays the process of setting up a grid controller within XgridLite. To start the Xgrid controller service, simply click the "Start" button. Authentication settings are configured with the "Set Client Password..." and "Set Agent Password..." buttons. From the XgridLite panel, you can also reset the controller or turn it off.


XgridLite

Show All

Xgrid Controller Off

Start — Click Start to activate the Xgrid controller. When active, the controller can accept jobs from clients and distribute them to available agents.

☑ Require password from clients — Set Client Password...

☑ Provide password to agents — Set Agent Password...

Reset — Click Reset to stop the controller and remove all database files. All clients will be disconnected, and all jobs will be lost. This operation cannot be undone.

Support Free and Open-Source Software Development

View License / Donate — XgridLite is free and open-source software, released the GNU General Public License, Version 2, allowing you to freely modify and redistribute the software under the terms of the license. You can encourage my open-source development efforts by donating money.

Click the lock to prevent further changes.

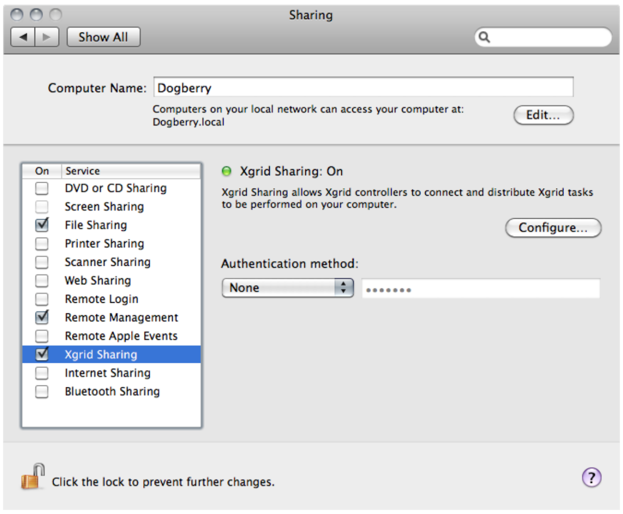
XgridLite 1.1.1
Copyright ©2005-2010 Ed Baskerville.
All rights reserved.

Figure 8: Setting up a controller

Unlike many other client-server relationships, the Xgrid controller authenticates to the client, rather than the other way around. That is, the password entered in the XgridLite "Set Agent Password..." field is *provided* to agents, not required of them. Individual agents must be configured to require that particular password (or none at all) in order to join the Xgrid.

Setting up agents

To set up an agent to join the Xgrid, open the Sharing preferences pane ("Internet & Wireless" section in OS 10.6) in System Preferences (see Figures 9 and 10). Select the "Xgrid Sharing" item in the list on the left. Click the "Configure..." button. From here, you can choose whether the agent should join the first available Xgrid, or a specific one. The latter of these options is usually best – the drop-down menu of controllers will auto-populate with all that are currently running.



Computer Name: Dogberry

Computers on your local network can access your computer at: Dogberry.local

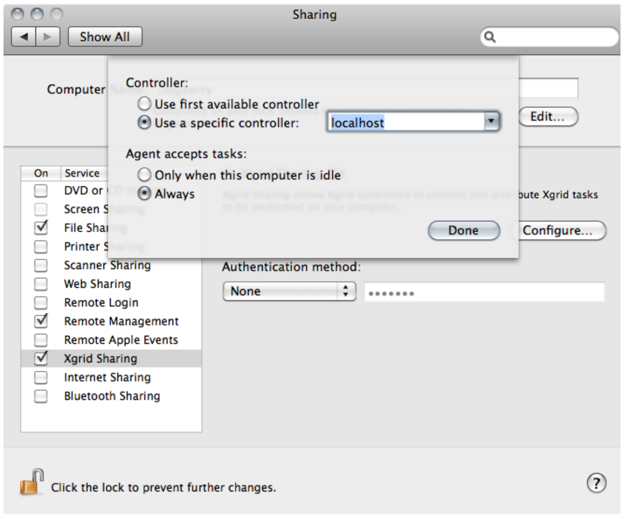
On	Service
☐	DVD or CD Sharing
☐	Screen Sharing
✓	File Sharing
☐	Printer Sharing
☐	Scanner Sharing
☐	Web Sharing
☐	Remote Login
✓	Remote Management
☐	Remote Apple Events
✓	Xgrid Sharing
☐	Internet Sharing
☐	Bluetooth Sharing

● Xgrid Sharing: On

Xgrid Sharing allows Xgrid controllers to connect and distribute Xgrid tasks to be performed on your computer.

Authentication method: None •••••••

Figure 9: Sharing setup to configure an agent for Xgrid



Controller:
○ Use first available controller
◉ Use a specific controller: localhost

Agent accepts tasks:
○ Only when this computer is idle
◉ Always

On	Service
☐	DVD or CD Sharing
☐	Screen Sharing
✓	File Sharing
☐	Printer Sharing
☐	Scanner Sharing
☐	Web Sharing
☐	Remote Login
✓	Remote Management
☐	Remote Apple Events
✓	Xgrid Sharing
☐	Internet Sharing
☐	Bluetooth Sharing

Authentication method: None •••••••

Figure 10: Specifying the controller for an agent

If there are multiple controllers on a network, you may want to choose one specifically so that the agent does not default to a different one. You may also choose whether or not the agent should accept tasks when idle. Then click “Done” and choose the desired authentication method (most likely “Password” or [less advisable] “None”) in the main window. If applicable, enter the password that will be required of the controller. Then, select the checkbox next to the “Xgrid Sharing” item on the left. The agent needs to be restarted to complete the process.

Note that a controller can also be an agent within its grid. However, it is important to consider that if the controller is also an agent, this has the potential to slow a simulation.

Automation

For those with more experience in system administration and access to Apple Remote Desktop (ARD) or secure shell (SSH) on the agent machines (or work with an administrator who does), the process of setting up agents can be automated. R can be installed remotely (allowing you to be sure that it is installed in the same location on each machine), the file that contains Xgrid agent settings can be pushed to all agents and, finally, the agent service can be activated. We will not discuss the ARD remote package installation process in detail here, but a full explanation can be found in the Apple Remote Desktop Administrator Guide at `http://images.apple.com/remotedesktop/pdf/ARD_Admin_Guide_v3.3.pdf`.

The settings for the Xgrid agent service are stored in the plist file located at ‘/Library/Preferences/com.apple.xgrid.agent.plist’. These settings include authentication options, controller binding, and the number of physical processor cores the agent should report to the controller (see the note below for a discussion of this value, which is significant). The simplest way to automate the creation of an Apple Xgrid is to configure the agent service on a specific computer with the desired settings and then push the above plist file to the same relative location on all agents. Keep the processor core settings in mind; you may need to push different versions of the file to different agents in a heterogeneous grid (see Note 3 below). Once all agents have the proper plist file, run the commands ‘`xgridctl agent enable`’ and ‘`xgridctl agent on`’ on the agents.

Notes

1. Mac OS X Server provides several additional options for setting up an Xgrid controller. These include Kerberos authentication and customized Xgrid cache files location. Depending on your network setup, these features may bolster security, but we will not explore them further here.

2. R **must** be installed in the same location on each agent. If it is installed locally by an end-user, it should default to the proper location, but it is worth verifying this before running tasks, otherwise they may fail. As mentioned previously, installing R remotely using ARD is an easy way to ensure an identical instantiation on each agent.

3. The reported number of processor cores (the `ProcessorCount` attribute) is significant because the controller considers it the maximum number of tasks the agent can execute simultaneously. On Mac OS X 10.6, agents by default report the number of cores (not processors) that they have (see discussion at `http://lists.`

apple.com/archives/Xgrid-users/2009/Oct/msg00001.html). The ProcessorCount attribute can be modified depending on the nature of the grid. Since R uses only 1 processor per core, this may leave many processors idle if the default behavior is not modified.

4. Because Xgrid jobs are executed as the user "nobody" on grid agents, they are given lower priority if the CPU is not idle.

Acknowledgements

Thanks to Tony Caldanaro and Randall Pruim for technical assistance, as well as Molly Johnson, two anonymous reviewers and the associate editor for comments and useful suggestions on an earlier draft. This material is based in part upon work supported by the National Institute of Mental Health (5R01MH087786-02) and the US National Science Foundation (DUE-0920350, DMS-0721661, and DMS-0602110).

Bibliography

Apple Inc. *Mac OS X Server: Xgrid Administration and High Performance Computing (Version 10.6 Snow Leopard)*. Apple Inc., 2009.

L. M. Collins and S. T. Lanza. *Latent Class and Latent Transition Analysis: With Applications in the Social, Behavioral, and Health Sciences*. Wiley, 2009.

M. J. Denwood. *runjags: Run Bayesian MCMC Models in the BUGS syntax from Within R*, 2010. URL http://cran.r-project.org/web/packages/runjags/. R package version 0.9.9-1.

A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994. URL http://www.csm.ornl.gov/pvm/.

T. Hesterberg. It's time to retire the $n \geq 30$ rule. *Proceedings of the Joint Statistical Meetings*, 2008. URL http://home.comcast.net/~timhesterberg/articles/.

N. Horton and S. Anoke. *xgrid: Access Apple Xgrid using R*, 2012. URL http://CRAN.R-project.org/package=xgrid. R package version 0.2-5.

P. K. Keel, M. Fichter, N. Quadflieg, C. M. Bulik, M. G. Baxter, L. Thornton, K. A. Halmi, A. S. Kaplan, M. Strober, D. B. Woodside, S. J. Crow, J. E. Mitchell, A. Rotondo, M. Mauri, G. Cassano, J. Treasure, D. Goldman, W. H. Berrettini, and W. H. Kaye. Application of a latent class analysis to empirically define eating disorder phenotypes. *Archives of General Psychiatry*, 61(2):192–200, February 2004.

N. M. Li and A. J. Rossini. rpvm: Cluster statistical computing in R. *R News*, 1(3):4–7, 2001.

D. A. Linzer and J. B. Lewis. poLCA: An R package for polytomous variable latent class analysis. *Journal of Statistical Software*, 42(10):1–29, 2011. URL http://www.jstatsoft.org/v42/i10/.

A. J. Rossini, L. Tierney, and N. Li. Simple parallel statistical computing in R. *Journal of Computational and Graphical Statistics*, 16(2):399–420, 2007.

S. S. Sawiloswky and R. C. Blair. A more realistic look at the robustness and Type II error properties of the t test to departures from population normality. *Psychological Bulletin*, 111(2):352–360, 1992.

S. A. Swanson, K. Lindenberg, S. Bauer, and R. D. Crosby. A Monte Carlo investigation of factors influencing latent class analysis: An application to eating disorder research. *Journal of Abnormal Psychology*, 121(1):225–231, 2011.

S. Urbanek. *multicore: Parallel processing of R code on machines with multiple cores or CPUs*, 2011. URL http://CRAN.R-project.org/package=multicore. R package version 0.1-7.

D. Wegener, T. Sengstag, S. Sfakianakis, and A. A. S. Rüping. GridR: An R-based grid-enabled tool for data analysis in ACGT Clinico-Genomic Trials. *Proceedings of the 3rd International Conference on e-Science and Grid Computing (eScience 2007)*, 2007.

B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice Hall, 1999.

H. Yu. Rmpi: Parallel statistical computing in R. *R News*, 2(2):10–14, 2002.

Sarah Anoke
Department of Mathematics and Statistics, Smith College
Clark Science Center, 44 College Lane
Northampton, MA 01063-0001 USA
sanoke527@gmail.com

Yuting Zhao
Department of Mathematics and Statistics, Smith College
Clark Science Center, 44 College Lane
Northampton, MA 01063-0001 USA
yuzhao@smith.edu

Rafael Jaeger
Brown University
Providence, RI 02912 USA
rafael_jaeger@brown.edu

Nicholas J. Horton
Department of Mathematics and Statistics, Smith College
Clark Science Center, 44 College Lane
Northampton, MA 01063-0001 USA
nhorton@smith.edu