Example 1: Using an Apple Xgrid to Assess the Robustness of the One-Sample t-test

Sarah C. Anoke, Yuting Zhao, Nicholas J. Horton*
Department of Mathematics and Statistics
Smith College

July 15, 2011

Contents

1 Introduction

Many scientific computations can be sped up by dividing them into smaller tasks and distributing the computations to multiple systems for simultaneous processing. Such a process is referred to as *parallel computing*. When performed on existing grids of computers, this method can dramatically increase computation speed. Several solutions exist to facilitate this type of computation within R, and we describe one such solution here, that involves using the Apple Xgrid (Apple, 2009), a parallel computing environment.

We created the **xgrid** package to provide a simple interface to this distributed computing system (Horton et al., 2011). The package facilitates use of an Apple Xgrid for distributed processing of a job with many independent repetitions, by simplifying task submission (or *gridstuffing*) and collation of results. We demonstrate use of our package in the context of a real, although relatively simple, statistical problem.

*Corresponding author: nhorton@smith.edu

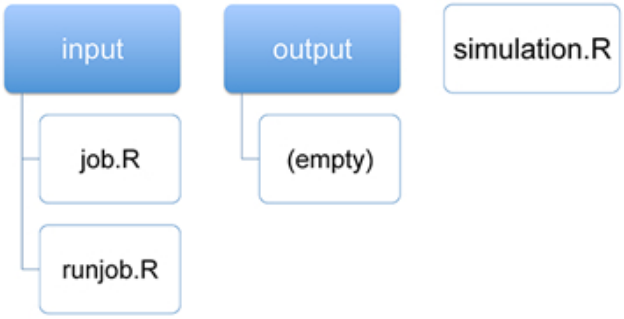

Figure 1: File structure to access the grid

2 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. This particular study runs very quickly as a loop in R, even for a large number of simulations, and as a result the use of the Xgrid actually slows down the calculation. However, we provide it for pedagogical reasons as a simple example to help describe the setup and test that the Xgrid is functioning appropriately.

3 Using the Grid for a Simulation Study

3.1 Directory Structure

Our first step is to set up an appropriate directory structure for our simulation (Figure 1).

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', defines the code to run a particular task (Figure 2).

For 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 the function `job()` returns a data frame with the rejection results and the corresponding sample size.

The folder 'input' also contains 'runjob.R', which retrieves and stores command line arguments from the controller, and passes them to `job()` (Figure 3). 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 needed for the simulation. In this example, none are needed.

```
# Assess the robustness of the one-sample
# t-test when underlying data are exponential
# this function returns a dataframe 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'

```
source("job.R")
# processargs expects 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
processargs = function() {
   args = commandArgs(trailingOnly=TRUE)
   return(list(ntask=as.numeric(args[1]),
      param=args[2], resfile=args[3]))
}
arg = processargs()
res0 = job(arg$ntask, param=arg$param)
# stash the results
save(res0, file=arg$resfile)
```

Figure 3: Contents of 'runjob.R'

```
library(xgrid)
# run the simulation
res = xgrid(Rcmd="runjob.R", param=30,
   numsim=10000, ntask=1000, verbose=FALSE)
# analyze the results
with(res, table(leftreject,rightreject))
```

Figure 4: Contents of 'simulation.R'

```
> list.files("output")
 [1] "RESULT-10000"              "RESULT-10001"              "RESULT-10002"
 [4] "RESULT-10003"              "RESULT-10004"              "RESULT-10005"
 [7] "RESULT-10006"              "RESULT-10007"              "RESULT-10008"
[10] "RESULT-10009"              "runjob.RRESULT-10000.Rout" "runjob.RRESULT-10001.Rout"
[13] "runjob.RRESULT-10002.Rout" "runjob.RRESULT-10003.Rout" "runjob.RRESULT-10004.Rout"
[16] "runjob.RRESULT-10005.Rout" "runjob.RRESULT-10006.Rout" "runjob.RRESULT-10007.Rout"
[19] "runjob.RRESULT-10008.Rout" "runjob.RRESULT-10009.Rout"
```

Figure 5: Contents of 'output' directory

The next item in the directory structure is the folder 'output', which will contain the results from the simulations. If it doesn't exist, the **xgrid** package will create it. The 'output' directory has a complete listing of the individual results as well as the R output from the remote agents. This can be useful for debugging in case of problems.

The final item in the directory structure is 'simulation.R', which contains an R script to be run on the client machine that calls `xgrid()` from the package **xgrid** (Figure **??**). This function submits the simulation to the grid for calculation. Results from all jobs are returned as one object, `res`. The call to `with()` summarizes all results in a table and prints them to the console.

Here we specify a total of 10,000 simulations, to be split into 10 jobs of 1,000 simulations each. Note that the number of jobs is calculated as the total number of simulations (`numsim`) divided by the number of tasks per job (`ntask`). Each simulation has a sample size of `param`.

Jobs are submitted to the grid by running 'simulation.R'. After the completion of each job, the results are saved to a file in the 'output' directory. Figure 4 is an example of what would be seen in the 'output' folder. As expected, there are ten files of the form 'RESULT-1000#', which contain the results from each individual job. The second set of ten files (e.g. 'runjob.RRESULT-1000#.Rout') contain the code that was run to generate the corresponding result file.

3.2 Retrieval and Analysis of Results

After the completion of the entire simulation study, results from all `numsim` simulations are collated and returned by the `xgrid()` function as the object `res`. This object is also saved as the file 'RESULTS.rda', at the top of the directory structure (see Figure 5).

In this case, `res` is a data frame with `numsim` rows (one row for each simulation) and three columns (as defined in the `return` statement of 'job.R'). In Figure 6, we list the dimensions of `res` and summarize the results using `with()`.

```
> list.files()
[1] "RESULTS.rda"  "input"        "output"       "simulation.R"
> list.files("input")
[1] "job.R"    "runjob.R"
```

Figure 6: Contents at the top of the directory structure, and of 'input'

```
> dim(res)
[1] 10000     3
> with(res, table(leftreject,rightreject))
          rightreject
leftreject FALSE TRUE
     FALSE  9299   69
     TRUE    632    0
```

Figure 7: Dimensions of `res` and analysis using `with()`

In terms of our motivating example, when the underlying data are normally distributed, we could 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 6.3% and 0.7% for the 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.

4 Acknowledgement

This material is based in part upon work supported by the National Institute of Mental Health (5R01MH087786-02) and the US National Science Foundation (0920350 and 0721661).

5 Bibliography

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

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

N. J. Horton, S. C. Anoke, Y. Zhao, and H. Jaeger. Xgrid and R: Parallel distributed processing using heterogeneous groups of apple computers. *In revision*, 2011.

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.