R at the U of R: Benchmarking Computing Resources

Jonathan P. Olmsted

Contents

1 Introduction

1.1 Description of Document

This document has been prepared by the author at the request of the Director of **the star lab**, Curtis Signorino, and the Political Science Department System Administrator, Tony Lenzo. Various programs typical of the analysis performed by political scientists are run in the `R` statistical environment (or derivative projects).[1] These tests are performed using several numerical libraries, under multiple operating systems, within different implementations of the `R` environment, and using a range in the number of processors dedicated to the tests. All of this is aimed at informing users and administrators about the relative performance of computing resources at the University of Rochester.

The author prepared this at the beginning of his tenure as **the star lab** Fellow. Consequently, **the star lab** provided support for the preparation of this document. Information about **the star lab** can be found at `www.rochester.edu/college/psc/thestarlab/main/index.php`. Contact information for the **the star lab** Director, Political Science System Administrator, and current **the star lab** Fellow is listed on the website.

[1] `R` is an open-source language and environment for statistical computing. More information can be found at `cran.r-project.org`

Jonathan P. Olmsted
jpolmsted@gmail.com

The primary purpose of this document is to provide technical details about the scope and implementation of the benchmarking project and to act as a supplement to the executive summary. A copy of the executive summary can be obtained at `http://www.rochester.edu/college/gradstudents/jolmsted/computing/` or by contacting the author. Results are not included in this report, but are in the executive summary document. In general, the executive summary should be read before using this document.

The remainder of the document is structured into two main remaining parts. First, there is a section describing the scope of the benchmarking procedure. This includes the various hardware, software, and test configurations. Although this information is in the executive summary, it is presented here in more detail. Second, there is a section documenting the use of the source code for the project.

2 Scope of the Benchmarking

This section summarizes the environments in which the benchmarking occurs (Section 2.1) and the procedures used to conduct the benchmarking (Section 2.2).

2.1 Environments

In this section the various hardware configurations (Section 2.1.1), operating systems (Section 2.1.2), software environments (Section 2.1.3), numerical libraries (Section 2.1.4), and number of processes (Section 2.1.5) used in the benchmarking are described.

2.1.1 Machine Setup

This section provides a brief outline of the various machines used in the benchmarking. Some differences in hardware are irrelevant for certain tests, but the focus in others. However, whenever possible, performance under certain conditions will be compared to the performance under other conditions on the same machine. Then, to whatever extent possible auxillary conditions will be held constant so that machines can be compared.

Machine Setup 1—HP dv6000

- Notebook
- AMD Turion Processor — 2 cores $\times$ 2.0 GHz
- 2 GB RAM

Machine Setup 2—Dell P690

- Desktop
- Intel Xeon Processor — 8 cores $\times$ 2.66 GHz
- 3.25 GB RAM

Machine Setup 3—Mars Cluster 1

- 22 Node Cluster, Master is Mars1[2]
- Mars20 — not present
- Mars2, Mars7-9, Mars13-15, Mars17-18, Mars21
  - 2 cores $\times$ 3.4 GHz
  - 2 GB RAM
- Mars3-6, Mars10-12, Mars16, Mars19
  - 1 cores $\times$ 3.4 GHz

[2]In the executive summary, "Mars1" refers to a version of the cluster and not a specific node on the cluster. Here it refers to the master node.

Jonathan P. Olmsted
jpolmsted@gmail.com

  - 2 GB RAM
- Mars22, Mars23
  - 4 cores × 3.4 GHz
  - 2 GB RAM

Machine Setup 4—BlueHive Cluster

The Bluehive Cluster is sufficiently large and powerful that for a full description of the hardware one should visit `https://www.rochester.edu/its/web/wiki/crc/index.php/BlueHive_Cluster`.

- 120 Node Cluster
- 848 cores
- Each Node has 18 GB of local high performance HDD
- Each Node has at least 4 GB of RAM

Machine Setup 5—Mars Cluster 2

- 24 Node Cluster, Master is Mars1
- 4 cores × 3.0 GHz
- 4 GB RAM

Each of the different software environments and numerical libraries do not exist on each of the various machines. That is, the benchmarking project is not set up as a complete factorial design where each OS is tested with each numerical library, for example. To indicate which of the remaining environmental factors are tested on a given machine the description of the factors will be followed by a **M1**, **M2**, **M3**, **M4**, or **M5**—indicating the machines on which they are tested.

2.1.2 Operating System

Operating systems are listed for completeness. Among the *nix operating systems, performance should not be expected to vary, all else equal.

Windows XP SP3

- Official: `http://www.microsoft.com/windows/windows-xp/default.aspx`
- Wikipedia: `http://en.wikipedia.org/wiki/Windows_xp`
- Machines: **M1, M2**

Ubuntu 10.04 LTS

- Official: `http://www.ubuntu.com/`
- Wikipedia: `http://en.wikipedia.org/wiki/Ubuntu_%28operating_system%29`
- Machines: **M1**

Fedora Core 6

- Official: `http://fedoraproject.org/`
- Wikipedia: `http://en.wikipedia.org/wiki/Fedora_core_6`
- Machines: **M3**, **M5**

Red Hat Enterprise Linux 5.5

- Official: `http://www.redhat.com/rhel/`
- Wikipedia: `http://en.wikipedia.org/wiki/Rhel`
- Machines: **M4**

Jonathan P. Olmsted
jpolmsted@gmail.com

2.1.3 Software Environment

CRAN `R`

- Official: `http://www.r-project.org/`
- Wikipedia: `http://en.wikipedia.org/wiki/CRAN_%28R_programming_language%29#CRAN`
- Versions:

  R 2.11.1: M1, M2, M4, M5
  R 2.9.2: M3

Revolution `R`

- Official: `http://www.revolutionanalytics.com/products/revolution-enterprise.php`
- Wikipedia: `http://en.wikipedia.org/wiki/CRAN_%28R_programming_language%29#CRAN`
- Versions:

  Revo R Enterprise 4.0: M1, M2

2.1.4 Numerical Libraries

Linear algebra operations are often calculated outside of `R` by subroutines contained in a set of Basic Linear Algebra Subroutines (BLAS). One way of improving the speed of `R` is to tell `R` to use a special **BLAS**. This is typically done when `R` is compiled from source. Instructions on how to do this can be found at `http://cran.r-project.org/doc/manuals/R-admin.html#Linear-algebra` or `http://www.rochester.edu/college/gradstudents/jolmsted/computing/`.

`R` **Default Internal Reference BLAS:**

- `R` comes with its own BLAS which is built in to the binary file. This is the default setup. Information for the reference BLAS is located at `http://www.netlib.org/`.
- Machines: **M1, M2**

`R` **Default Shared BLAS:**

- `R` without using an alternative **BLAS**, the default **BLAS** can be stored externally. Then, every call to the these sets of subroutines flows just through this file. Changing the **BLAS** requires swapping out this file and not re-compilation. Information for the reference BLAS is located at `http://www.netlib.org/`.
- Machines: **M1, M3, M5**

GotoBLAS:

- Information about the performance **BLAS** GotoBLAS can be found at `http://www.tacc.utexas.edu/resources/software/`. Registration is required for use. GotoBLAS can be compiled as either a single-threaded library or a multi-threaded library.
- Machines: **M1**

ATLAS:

- **ATLAS** is another performance set of subroutines. It automatically configures itself based on performance tests when being built on each machine. Information can be found at `http://math-atlas.sourceforge.`
- Machines: **M4**

Intel MKL:

Jonathan P. Olmsted
jpolmsted@gmail.com

- The **Intel MKL** set of subroutines come with a more restricted license than either of the other two high-performance libraries listed above. It is included because **Revolution Computing**'s version of `R` utilizes this library. They secured a special agreement to distribute this library with their version of `R`. See `http://software.intel.com/en-us/intel-mkl/`. Also, see `http://www.revolutionanalytics.com/why-revolution-r/which-r-is-right-for-me.php`. The optimized libraries referred to in the **Revolution** packagings are **Intel MKL**.
- Machines: **M1, M2, M4**

2.1.5 Number of Processes

Each unique combination of hardware, operating system, `R` version, and numerical library considered will be distributed to 1, 2, 4, and 8 tasks whenever possible. On **M1**, a machine where this is not possible, only 1 and 2 tasks are considered.

2.2 Types of Tests

The tests used in this benchmarking project come from the package `SLbench`. More information on them can be found at `http://www.rochester.edu/college/gradstudents/jolmsted/computing/SLbench/` or on `CRAN`. Below, I briefly describe the tests used.

2.2.1 Random Number Generation

Many draws are taken from simple probability distributions and discarded.

2.2.2 Optimization

A Poisson statistical model of US agency creation is estimated in the Maximum Likelihood framework using `optim()`.

2.2.3 Matrix Operations

Several simple operations are performed on some small matrices.

2.2.4 Ideal Point Estimation

The `wnominate` package is used to estimate ideal points for senators from the 90th US Senate. Five bootstrap iterations are used and the spatial model is restricted to two dimensions.

2.2.5 Standard Benchmark Script

A standard benchmarking script from the `R` community is used. This script performs more complicated matrix operations, Fast Fourier Transforms, sorting of numbers, and linear regression.

3 Benchmark Procedure

The code associated with the benchmarking project is by no means portable. The only real common, environment-independent code is the code associated with the `SLbench` package, which provides the various tests. Because of this lack of portability, there is no value in distributing all of the source code for how the benchmarks were run on each different platform. Instead, I provide and discuss a self-contained subset of the tests that were run on **M1**.

After this discussion, however, how these files would be used as templates for someone running similar analyses on other machines should be evident.

Jonathan P. Olmsted
jpolmsted@gmail.com

3.1 Code Structure

As I reference files, I assume that the source archive distributed with this document is located at `path/to/src`. First, I briefly describe the three different kinds of files used in this project. Second, I verbosely walk through an example of each.

3.1.1 Overview

There are different kinds of files.

Shell Call Files These files are located in the subdirectories of `path/to/src/sh/` beginning with "`test`". A dummy template file is located at `path/to/src/sh/_test.script` and can be used to create new files. The user sets environmental variables (in Linux or in Windows). These variables are specific to the environment being evaluated. A new file is required for each new combination of environmental settings (à la Section 2.1). However, new shell call files aren't needed for each test in Section 2.2, though. When the script is run, the logic of it is as follows. Variables are defined in the script.[3] Then, the script passes the remainder of the job off to a Shell Common File which then uses the variable definitions

Shell Common Files These files are located in the subdirectory `path/to/src/sh/common/`. There is a unique shell common file for each different platform. Platforms are considered distinct if they require different commands to submit an `R` file for batch evaluation. For example, the command to invoke `R` on a Windows machine and a Linux machine will look different, so they require different shell common files. However, the Linux cluster and the Linux desktops in this project require different shell common files, too, because the details of running `R` vary. These files simply contain the five different `R` submissions that correspond to each of the five tests in Section 2.2.

R Test Call Files These files (located in the subdirectory `path/to/src/R/calls/` are not OS scripting files, but are `R` files. There is one per test from Section 2.2. These files read the environmental settings from the variables and perform the necessary tasks accordingly. There are only five of these and they do not need to be changed.

3.1.2 Example

Now, I walk through one complete example that illustrates how the scripts work together and where changes would need to be made if a similar project were run on another machine.

Shell Call File. See the example `test.sh` file. This file has two parts. First, it sets Bash environmental variables for my Linux laptop. Second, it calls another shell script to handle the rest of the execution after the variables are set.

The environmental variables in the list are of two types. There are "switches" which have effects and there are "notes" which have their values saved for reference. For example, the environmental variable `srcdir` is a switch. The value of this variable is used by `R` to know where it can find various source files. If this is set incorrectly an error will ensue. On the other hand, `machine` is a note. The `R` output of this test (whose arbitrary identifying number is 101) will contain an object that contains the contents of `machine` (a character vector). While the variable itself does not affect the execution of the code, knowing the machine on which the code was timed is necessary for the post-execution analysis. If this variable is accidentally set to the wrong value, it only affects labeling and record-keeping. In general, it should not result in any errors.

A description of the variables is as follows:[4]

`ID:` switch, affects filename of output; unique ID for job

[3] This is straightforward on *nix-based operating systems. On Windows, I use the powershell scripting environment which is sufficient for this purpose.

[4] These descriptions presume familiarity with filesystem locations, numerical libraries, and the parallelization code necessary to run `R` code in parallel on a Windows machine, a Linux desktop, and a Linux cluster. Discussion of this is far outside the purview of this technical report. These topics are covered in various tutorials on `http://www.rochester.edu/college/gradstudents/jolmsted/`.

Jonathan P. Olmsted
jpolmsted@gmail.com

`machine`: note, no effect, no error checking; which machine is the test run on?

`rbrand`: note, no effect, no error checking; which brand of `R` is used?

`rvers`: note, no effect, no error checking; which version of `R` is used?

`src`: note, no effect, no error checking; is `R` compiled from source?

`os`: note, no effect, no error checking; which OS is being used?

`numlib`: note, no effect, no error checking; which numerical library is begin used?

`srcdir`: switch, affects reading of code; where is the source directory on the filesystem?

`calldir`: switch, affects reading of code; where is the directory of `R` calls on the filesystem?

`outdir`: switch, affects saving of output; where is the directory of output on the filesystem?

`torque_N`: switch, affects parallelization; number of nodes to reserve through torque and number of processes to create in `R`, non-zero only when using torque.

`multicore_N`: switch, affects parallelization; number of processors to use with `multicore`, non-zero only when using `multicore`.

`smp_N`: switch, affects parallelization; number of processors to use on Windows with SMP, non-zero only when using SMP on Windows.

```
# !/bash

# ## BASH VARIABLES
export ID="101"
export machine="m1"
export rbrand="cran"
export rvers="2.11.1"
export src="T"
export os="ub"
export numlib="ref"
export srcdir="~/Documents/Academic/rochester/StarLab/Benchmarking/src/"
export calldir="~/Documents/Academic/rochester/StarLab/Benchmarking/src/R/calls/"
export outdir="~/Documents/Academic/rochester/StarLab/Benchmarking/data/"
export torque_N="0"
export multicore_N="1"
export smp_N="0"

# ## POWERSHELL VARIABLES
# $env:ID=""
# $env:machine=""
# $env:rbrand=""
# $env:rvers=""
# $env:src=""
# $env:os=""
# $env:numlib=""
# $env:srcdir='C:\Documents and Settings\olmjo\Benchmarking\src\'
# $env:calldir='C:\Documents and Settings\olmjo\Benchmarking\src\R\calls\'
# $env:outdir='C:\Documents and Settings\olmjo\Benchmarking\data\'
# $env:torque_N="0"
# $env:multicore_N="0"
# $env:smp_N="1"

## Linux Desktop
sh ../common/ld.sh

## Linux Cluster
## sh ../common/lc.sh

## Windows Desktop
## ..\common\wd.ps1

## Windows Desktop Revolution
## ..\common\wdr.ps1
```

../to_distribute/src/sh/test101/test.sh

Jonathan P. Olmsted
jpolmsted@gmail.com

This file would be run by running `sh test.sh` at the command line from within the directory `test101`. Running this shell script from within this directory is important since many of the paths referenced are relative. Once this script is run, the variables (explained above) are set and then another shell script is run. In the case of this run, the Linux desktop (hence, "ld") script is run.

Shell Common File. This very simple shell script depends on the system it is run on only insofar as it (1) presumes a certain binary is in the search path (i.e. `Rscript`) and (2) it presumes it is being called from within the proper directory given its relative references. As long as the shell call file is run from the proper directory, these path reference can be ignored. One important thing to note is that these `R` processes are being created in a shell environment where certain variables are already defined and so they have access to the switches and note from the previous step.

```
# !/bash

Rscript --vanilla ../../R/calls/callRNG.R
Rscript --vanilla ../../R/calls/callStandard.R
Rscript --vanilla ../../R/calls/callMatrix.R
Rscript --vanilla ../../R/calls/callIPE.R
Rscript --vanilla ../../R/calls/callOptim.R
```

../to_distribute/src/sh/common/ld.sh

There is a batch `R` command for each of the different tests in `ld.sh`. Here, we consider five different tests, each using the same environmental variable definitions and calling a different `R` script. I will discuss one of these next.

R Test Call File. The majority of the file `callIPE.R` is not specific to the ideal point test. First, environmental variables are read in from the shell environment and saved as a list. Second, any parallel backends are initialized based on the three parallelization environmental variables. Third, the package `SLbench` is loaded and the function `testIPE()` called. The execution of the 10 runs is timed. These times and the environmental variables are combined in a list in `R`. This list is then saved to the disk as an `R` file.

```
################################################################################
### FILENAME: callIPE.R
###
### AUTHOR: J. P. Olmsted --- jpolmsted@NOSPAM.gmail.com
###
### DATE: Tue Sep  7 14:15:56 2010
###
### LICENSE: GPL-2 See Below.
###
### DESCRIPTION:
################################################################################

## Benchmarking Script: IPE. Issues R Level Instructions for
## Benchmarking
## Copyright (C) 2010 J. P. Olmsted

## This program is free software; you can redistribute it and/or modify
## it under the terms of the GNU General Public License as published by
## the Free Software Foundation; either version 2 of the License, or (at
## your option) any later version.

## This program is distributed in the hope that it will be useful, but
## WITHOUT ANY WARRANTY; without even the implied warranty of
## MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
## General Public License for more details.

## You should have received a copy of the GNU General Public License
## along with this program; if not, write to the Free Software
## Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
## 02110-1301 USA.

## For any problems or suggestions, contact:
##   J. P. Olmsted --- jpolmsted@NOSPAM.gmail.com

## Full text:
##   http://www.gnu.org/licenses/gpl-2.0.txt

################################################################################

```


Jonathan P. Olmsted
jpolmsted@gmail.com

```
## #################
## Read in Arguments
## #################

lArgs <- as.list(Sys.getenv(c("ID",
                              "machine",
                              "rbrand",
                              "rvers",
                              "src",
                              "os",
                              "numlib",
                              "srcdir",
                              "calldir",
                              "outdir",
                              "torque_N",
                              "multicore_N",
                              "smp_N"
                              )
                            )
                 )

## ##############
## Hack Arguments
## ##############

lArgs$torque_N <- as.integer(lArgs$torque_N)
lArgs$multicore_N <- as.integer(lArgs$multicore_N)
lArgs$smp_N <- as.integer(lArgs$smp_N)

## #################
## Cluster Specific
## #################

if (lArgs$torque_N > 0)
  {
    library(snow)
    library(doSNOW)

    nNodes <- as.integer(lArgs$torque_N)
    clType <- "MPI"
    cl<- makeCluster(nNodes, clType)
    lArgs$clType <- clType

    registerDoSNOW(cl)
  }

## ##############
## Linux Specific
## ##############

if (lArgs$multicore_N > 0)
  {
    library(SLbench)
    library(multicore)
    library(doMC)

    registerDoMC(lArgs$multicore_N)
  }

## #################
## Windows Specific
## #################

if (lArgs$smp_N > 0)
  {
    library(SLbench)
    library(doSMP)
    rmSessions(all.names = TRUE)

    if (lArgs$numlib == "sh_intel1")
      {
        setMKLthreads(1)
      }
    if (lArgs$numlib == "sh_intel2")
      {
        setMKLthreads(2)
      }

    vSMP <- startWorkers(lArgs$smp_N)
    registerDoSMP(vSMP)
  }
```


Jonathan P. Olmsted
jpolmsted@gmail.com

```
121
122
123 ## ############
124 ## Cluster Run
125 ## ############
126 vTimes <- system.time(
127                       foreach(i = 1:10) %dopar% {
128                         library(SLbench)
129                         testIPE()
130                       }
131                       )
132
133 print(vTimes)
134
135 ## #################
136 ## Windows Specific
137 ## #################
138
139 if (lArgs$smp_N > 0)
140   {
141     stopWorkers(vSMP)
142   }
143
144
145 ## ######
146 ## Output
147 ## ######
148
149 output <- list(args=lArgs,
150                times=vTimes,
151                info=sessionInfo(),
152                test="IPE"
153                )
154
155 save(file=paste(
156          lArgs$outdir,
157          lArgs$ID,
158          "IPE.Rdata",
159          sep=""
160          ),
161      output
162      )
163
164 ################################################################################
```

../to_distribute/src/R/calls/callIPE.R

4 Summary

This technical report first describes the scope of the benchmarking project and second outlines the procedure. Although some of the details about the environments evaluated and the tests performed are contained in the executive summary, they are discussed in more detail here.

The code for this project is not portable, but the intuition behind it is. And, the code is easily adapted. Put succinctly, a script file is created for each environment to be evaluated. Each environment which shares a common call to the `R` binary file then uses the same shell common file. These shell common files call the same five `R` scripts for each environment evaluated. The `R` scripts dynamically read the variables, run the tests, and save the output.

Jonathan P. Olmsted
jpolmsted@gmail.com