III Seminário Internacional de Estatística com R
R for Science Integration Challenge
Niterói-RJ-Brasil - 22,23 e 24 de maio de 2018

ANÁLISE COMPARATIVA DE TEMPOS DE EXECUÇÃO DE CÓDIGO R INTERPRETADO E COMPILADO EM SISTEMAS WINDOWS E LINUX

Ciniro Aparecido Leite Nametala[1]

Gustavo Henrique Nunes[2]

Petrúcio Júnio do Amaral[3]

Resumo

Uma das principais características da linguagem de programação R é ser multiplataforma. Usuários de R, neste sentido, podem possuir interesse em conhecer as diferenças de desempenho da linguagem quando sendo executada sob sistemas operacionais diferentes. Essa questão é especialmente importante visto que R tem se consolidado com opção para atividades que envolvam análise de grandes massas de dados, modelos de aprendizado de máquina, processamento paralelo e afins. Neste contexto, é relatado neste artigo, um estudo que buscou averiguar o desempenho de R com RStudio, ambos instalados e operando sob sistemas operacionais Windows 10 e Linux com distribuição Ubuntu 17.10. O planejamento e projeto experimental envolveu diversas etapas visando diminuir ao máximo os possíveis ruídos de software e hardware que pudessem interferir na coleta de dados. Após a definição de um *dataset* de tempos de execução que levou em conta quatro cenários diferentes, avaliou-se as características das amostras e, com aplicação de testes de múltiplas comparações, pode-se constatar que R foi mais rápido quando usado sob Ubuntu 17.10 com código compilado.
Palavras-chave: Linguagem de Programação R, Sistemas Operacionais, Análise de Desempenho.

Abstract

One of the main features of the R programming language is to be cross-platform. R users in this sense may have an interest in knowing the differences in language performance when running under different operating systems. This issue is especially important because R has been consolidated with an option for activities involving analysis of large masses of data, machine learning models, parallel processing and others. In this context, it is reported in this article, a study that sought to ascertain the performance of R with RStudio, both installed and operated under Windows 10 and Linux in Ubuntu distribution 17.10. The planning and experimental design involved several steps aimed at minimizing possible software and hardware noise that could interfere with data collection. After defining a runtime dataset that took into account four different scenarios, the characteristics of the samples were evaluated and, with application of multiple comparison tests, it can be verified that R was faster when used under Ubuntu 17.10 with code compiled.
Keywords: Programming Language R, Operating Systems, Performance Analysis.

[1] Instituto Federal de Minas Gerais (IFMG-Campus Bambuí), ciniro.nametala@ifmg.edu.br

[2] Instituto Federal de Minas Gerais (IFMG-Campus Bambuí), gustavohenriquenunex@gmail.com

[3] Instituto Federal de Minas Gerais (IFMG-Campus Bambuí), petruciojunio@gmail.com

Introdução

Um dos interesses principais no uso da linguagem de programação R é que o código fonte produzido possa ser estruturado de forma a ser executado no menor tempo possível, ou seja, permita atingir os objetivos para os quais foi proposto levando-se em conta também o esforço computacional exigido pela ferramenta. Esse interesse pode ser considerado relevante ao se observar que mais de 17 mil tópicos sobre o tema “Desempenho da Linguagem R” estão hoje ativos em um dos principais fóruns de discussão sobre R na Internet, o *Nabble R Help* (R Community 2018).

Em muitas das vezes, a qualidade de desempenho de um código fonte está associada diretamente ao sistema operacional sob o qual este está sendo executado, nesse sentido, torna-se importante averiguar qual a magnitude desta influência. O experimento relatado a seguir trata da comparação de tempos de execução do R, em formato interpretado e compilado, sob os sistemas operacionais Windows na versão 10 e Linux com distribuição Ubuntu versão 17.10, ambos em arquitetura 64 bits. Buscou-se averiguar, em condições iguais para ambas as plataformas e formatos, qual ambiente favorece o melhor desempenho. As rotinas para condução dos experimentos foram implementadas utilizando-se a própria linguagem R.

Objetivo

Neste experimento buscou-se averiguar a variável “tempo de execução”, em segundos com precisão de duas casas decimais, pretendendo-se responder a seguinte pergunta de interesse:

> *Nos sistemas operacionais Ubuntu 17.10 e Windows 10, com execução de código interpretado e compilado, qual combinação de sistema operacional (ambiente) e tipo de código (formato) apresenta melhor desempenho utilizando-se linguagem R?*

Para isso foram realizadas coletas de tempo na execução de um mesmo código desenvolvido para a versão 3.4.4, quando em Windows 10, e versão 3.4.2, quando em Ubuntu 17.10. Ambos executados a partir da IDE RStudio 1.1.442. Todos os softwares utilizados no estudo trataram-se de sistemas originais e, quando foi o caso, com devido licenciamento, além de, também, serem as versões homologadas mais recentemente lançadas para cada plataforma, por suas respectivas fabricantes.

Os testes foram conduzidos, logo, levando-se em conta quatro cenários possíveis, sendo estes:

• Execução via RStudio de código R compilado sob sistema operacional Windows 10.

• Execução via RStudio de código R interpretado sob sistema operacional Windows 10.

• Execução via RStudio de código R compilado sob sistema operacional Ubuntu 17.10.

• Execução via RStudio de código R interpretado sob sistema operacional Ubuntu 17.10.

Com base nisto foi objetivo deste trabalho averiguar a possível influência destas configurações de ambiente quando, sob estes, executavam exatamente o mesmo código fonte.

Material e Método

A linguagem R por padrão é interpretada[4], entretanto quando se busca velocidade e não se tem por necessidade alterar o código a todo tempo, a compilação pode ser uma boa opção, visto que ao converter-se a linguagem R para uma linguagem intermediária (mais próxima da linguagem de máquina utilizada pelo sistema operacional) podem-se obter ganhos de desempenho. No R existem rotinas específicas para compilação de código em diversos ambientes alvo. Uma das opções para a compilação é o pacote *compiler* (TIERNEY, 2011), nativo do R desde a versão 2.14, usado para compilar arquivos, funções ou partes específicas do código. Caso o objetivo seja a realização de uma compilação completa, tem-se por opção também a utilização dos pacotes com foco em compilação *Just in Time*, como o *JIT* (MILBORROW, 2011) ou até o próprio pacote *compiler* com uso do método *enableJIT()*.

Estruturas mais primitivas de código como os mecanismos de *loops*, estruturas condicionais, termos de comparação e atribuição, tipagem implícita e explícita de dados, conversores de tipos e métodos de chamada e outros, geralmente apresentam melhores resultados quando compilados, pois, utilizam-se principalmente, nestes casos, dos recursos computacionais oferecidos pelo sistema operacional. Já a utilização de funções prontas e bibliotecas externas podem não responder tão bem a compilação, pois não se pode ter certeza da qualidade da sua arquitetura interna quando, por exemplo, esta for fechada e oriunda de terceiros. Outro ponto a se considerar é que a própria chamada de uma função

[4] Definições formais de linguagens interpretadas e compiladas podem ser vistas com maiores detalhes no trabalho de Scott (2009).

pronta já consome recursos que, em geral, são gerenciados, neste caso, pela própria linguagem de programação.

Assim, neste experimento optou-se pela utilização de um código base para os testes que privilegiasse, principalmente, a manipulação de variáveis tipadas visando testar a qualidade da alocação de memória provida pelo sistema operacional e, *loops* com cálculos, para averiguação do uso dos recursos de processador. Nenhum destes blocos de código depende de qualquer pacote externo.

Quatro funções foram desenvolvidas com base no código disponibilizado por Upadhyay (2015). O *script* combina métodos diversos de preenchimento de listas com e sem a utilização de funções internas. O tempo de execução de cada uma em versão interpretada (chamadas no experimento de f1, f2, f3 e f4) e compilada (chamadas no experimento de fc1, fc2, fc3 e fc4) foi aferido por meio do pacote *microbenchmark* (MERSMANN, 2014) levando-se em conta que, em cada uma, foi executado um *loop* de 30 mil iterações. O esforço computacional (consumo de recursos) em uma dada execução foi obtido por meio da função *Rprof()*, nativa do R, e posterior análise do arquivo de saída *out.out* com a função *proftable()* disponibilizada por Ross (2015). As funções do código base são comentadas a seguir e estão disponibilizadas publicamente para acesso no *link* da referência (NAMETALA, 2018).

• f1() e fc1(): Função que consome entre 55% e 60% dos recursos quando o código base é executado. Em comparação com as outras funções é a pior construção que pode ser realizada quando deseja-se utilizar listas dinamicamente preenchidas, pois aloca espaço na memória duas vezes a cada iteração quando é executada a sobrescrita da variável. Também é utilizado o método *append()* que adiciona o valor ao final da lista alterando o tamanho da mesma. O espaço na memória não é pré-alocado e a tipagem de dados é implícita.

• f2() e fc2(): Função que consome entre 35% e 40% dos recursos. Apesar de alterar em toda iteração o tamanho da lista, esta não sobrescreve a variável e não usa o método *append()*. Como na f1() e fc1() o espaço na memória não é pré-alocado e a tipagem de dados também é implícita.

• f3() e fc3(): Função que apresenta melhor performance entre as quatro consumindo menos que 1% dos recursos. A lista é pré-alocada na memória antes de seu preenchimento, fato que dispensa o redimensionamento. Outro ponto importante é a tipagem explícita, que garante a construção do tipo *list* apenas uma vez.

• f4() e fc4(): Função semelhante a f3() e fc3(), inclusive no consumo de recursos que é abaixo de 1%. A alocação de memória é feita previamente e a tipagem de dados também

é explícita, contudo ao invés de dois *loops* para preenchimento, esta utiliza a função *sapply()* que funciona como uma referência para a função *lapply()* que, por sua vez, aplica a atribuição em cada espaço da lista.

Coleta de Dados

Buscou-se inicialmente para o experimento tentar excluir todas as possíveis formas de ruído quando da execução do código base em ambas as plataformas. Assim, optou-se pela utilização das últimas versões lançadas dos sistemas operacionais, aplicadas à estas suas devidas e mais recentes atualizações, além da instalação apenas da linguagem R e da IDE RStudio. Os sistemas operacionais não foram instalados em computadores convencionais visto que, mesmo que fossem de iguais configurações, poderiam sofrer com diferenças construtivas. Para tanto foi utilizada a técnica de virtualização por meio do software Oracle VirtualBox, versão 5.2.8. A máquina *host* utilizada no experimento foi a de seguinte configuração: AMD Ryzen 5 1600 e memória RAM de 8 GB em Ubuntu 17.10 64 bits. Ambas as plataformas virtualizadas operaram exatamente na mesma configuração sendo: 4.096 MB de Memória RAM, alocação de 6 processadores, memória de vídeo de 128 MB, aceleração 3D VT-x/AMD-V com paginação aninhada, controladora única SATA (vdi) com alocação de 50 GB fixo e sem conexão à internet.

Importante ressaltar que a virtualização impede que os recursos alocados variem ao longo dos testes, entretanto apesar de diminuir, não elimina completamente ruídos inerentes que poderiam influenciar minimamente os resultados do experimento como temperatura do *hardware*, umidade, variações na tensão da rede elétrica e outros similares.

A coleta foi realizada em quatro lotes, uma para cada cenário. Antes de cada coleta a máquina virtual era encerrada e convertida para o estado da primeira inicialização, desprezando-se assim prováveis configurações automáticas do sistema operacional após a primeira execução atualizada. Nunca, em nenhuma dos momentos do experimento, tanto na máquina virtualizada quanto na máquina *host*, operaram-se outros softwares senão os unicamente utilizados para a coleta dos tempos. O protocolo para coleta foi executado para quatro amostras (uma em cada cenário) e em cada uma destas foram tomadas trinta observações, ou seja, trinta tempos de execução em cada situação envolvendo sistema operacional e formato. Estas observações iniciais foram utilizadas para estimar o tamanho amostral conforme descrito a seguir.

Análise da amostra inicial e definição de tamanho amostral

Os dados existentes na primeira amostra foram sumarizados para observação de indicadores simples como desvio padrão, média, mediana, valores máximos, mínimos e quartis. Neste primeiro momento não foi observada qualquer anomalia (como valores abusivamente distantes). Estas informações referentes as amostras iniciais podem ser visualizadas na Tabela 1.

Tabela 1 – Sumarização de medidas da amostra inicial.

Medida	**Windows 10**		**Ubuntu 17.10**	
	Compilado	**Interpretado**	**Compilado**	**Interpretado**
Mínimo	3.12	3.06	1.84	2.12
1º quartil	3.30	3.22	2.40	2.58
Mediana	3.39	3.32	2.52	2.70
Média	3.37	3.31	2.56	2.69
3º quartil	3.46	3.38	2.73	2.85
Máximo	3.58	3.56	3.06	3.20
Desvio padrão	0.12	0.12	0.26	0.22

Fonte: O AUTOR

Buscando-se averiguar a qualidade dos dados da amostra inicial, procedeu-se também a análise gráfica das observações. A princípio, pôde-se perceber com o gráfico *boxplot* da Figura 1 que os códigos executados sob sistema operacional Ubuntu 17.10 se mostraram mais rápidos frente aos executados em Windows 10. Quanto a premissa de normalidade, feita também via análise gráfica (cruzamento dos resíduos vs. uma normal teórica) e exibida na Figura 2, não foram detectados pontos que pudessem ser considerados *outliers*. A normalidade de cada amostra foi ainda avaliada pela aplicação do teste de Shapiro Wilk, com nível de significância de 0.05, que retornou p-valores de 0.36, 0.43, 0.30 e 0.97 para, respectivamente, cenários compilado e interpretado em Windows e, cenários compilado e interpretado em Ubuntu. Estes resultados, sugeriram, por consequência, que todas as amostras provinham de uma distribuição normal.

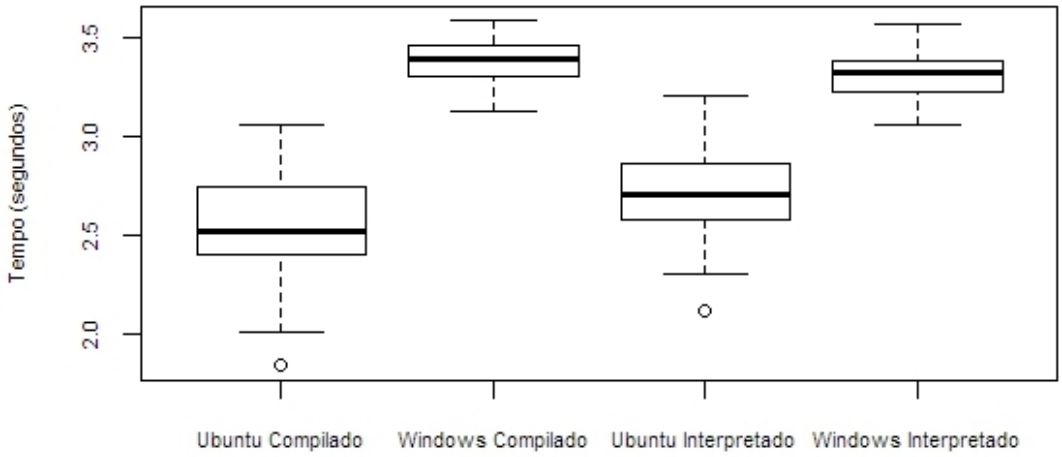

Figura 1 – *Boxplot* das amostras iniciais em cada cenário

Fonte: O AUTOR

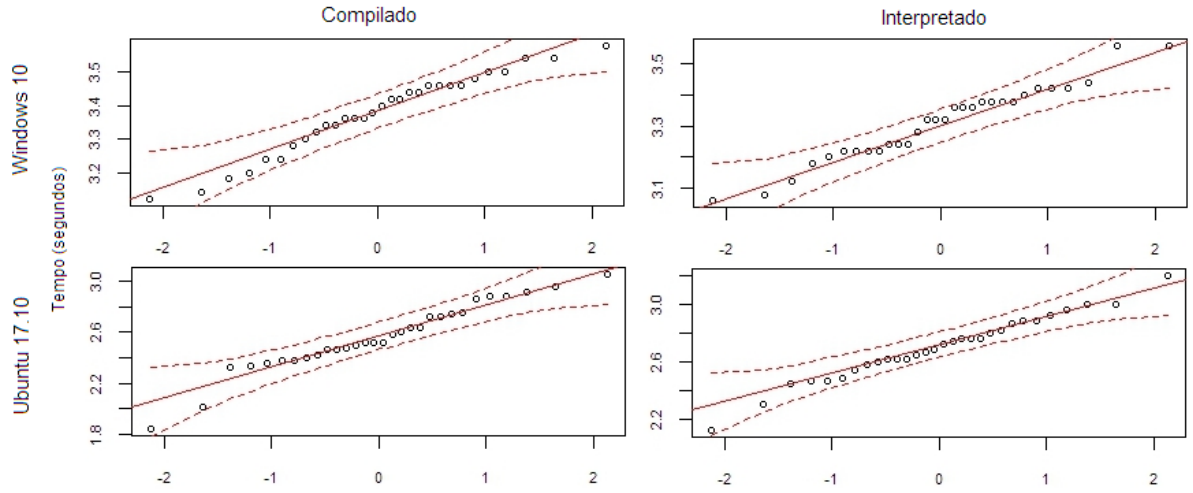

Figura 2 – Cruzamento dos resíduos das amostras iniciais vs. distribuição normal teórica.

Fonte: O AUTOR

Com a indicação de normalidade em todas as amostras iniciais, optou-se pelo uso do teste de potência ANOVA para definição do tamanho das amostras finais. Nesse sentido, visto que a variável analisada “tempo de execução” foi estipulada na unidade “segundos” e que, a amostra inicial apresentou valor mínimo de 1.84 e máximo de 3.58, definiu-se que o mínimo efeito de interesse prático δ^* a ser detectado pelo teste seria de 0.1. Como a coleta de dados pôde ser realizada sem custos, optou-se também pela utilização de uma potência alta, no caso, 0.8. O nível de significância do teste foi de $\alpha = 0.01$. A variância do vetor τ foi calculada com base na expressão que define dois níveis simetricamente em torno de uma média geral, assim utilizou-se a formulação da Equação 1 obtendo-se 0.00166.

$$\tau = \left\{ -\frac{\delta^*}{2}, \frac{\delta^*}{2}, 0, 0 \right\} \quad (1)$$

Para o desvio padrão $\hat{\sigma}$ foi executada uma média aritmética simples entre os desvios padrão das quatro amostras iniciais obtendo-se o valor de 0.0381. Os resultados, exibidos na Listagem 1, indicaram a necessidade de aproximadamente 120 observações para cada um dos quatro grupos. Utilizando-se então o mesmo protocolo de coleta das amostras iniciais procedeu-se a coleta das amostras definitivas.

Listagem 2 – Resultados do teste de tamanho amostral

```
Balanced one-way AOV power calculation      within.var = 0.03818106
groups = 4                                  sig.level = 0.01
n = 119.4611                                power = 0.8
between.var = 0.001666667
```

Resultados e Discussão

A coleta da amostra definitiva tratou da aferição de 480 observações, 120 em cada cenário. Após esta ter sido realizada, executou-se a sumarização dos dados que pode ser vista na Tabela 2. Adicionalmente, na Figura 3 apresenta-se o gráfico *boxplot* destes valores.

Como pode-se perceber existe semelhança entre a amostra inicial e a amostra final, sugerindo, mais uma vez, que códigos executados em plataforma Ubuntu são mais rápidos que os executados sob Windows. No entanto, no interesse de constatar definitivamente a existência de possíveis diferenças entre as médias de “tempo de execução” em cada grupo e, levando-se em conta a sugestão de normalidade observada nas amostras iniciais, pretendeu-se aplicar, a princípio, o teste de ANOVA paramétrica para realização das comparações. Para tanto, na sequência, foram avaliadas as premissas exigidas por este teste.

Tabela 2 – Sumarização de medidas da amostra final.

Medida	Windows 10		Ubuntu 17.10	
	Compilado	Interpretado	Compilado	Interpretado
Mínimo	3.02	3.06	1.78	1.76
1º quartil	3.12	3.18	2.48	2.62
Mediana	3.16	3.32	2.62	2.80
Média	3.16	3.30	2.59	2.78
3º quartil	3.20	3.40	2.74	2.96

Máximo	3.30	3.60	3.04	3.34
Desvio padrão	0.05	0.12	0.21	0.24

Fonte: O AUTOR

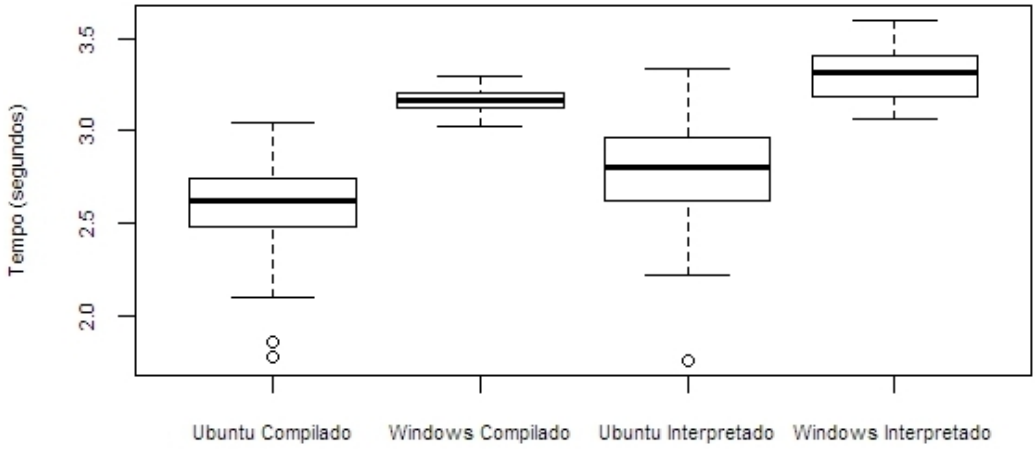

Figura 3 – *Boxplot* das amostras finais em cada cenário.

Fonte: O AUTOR

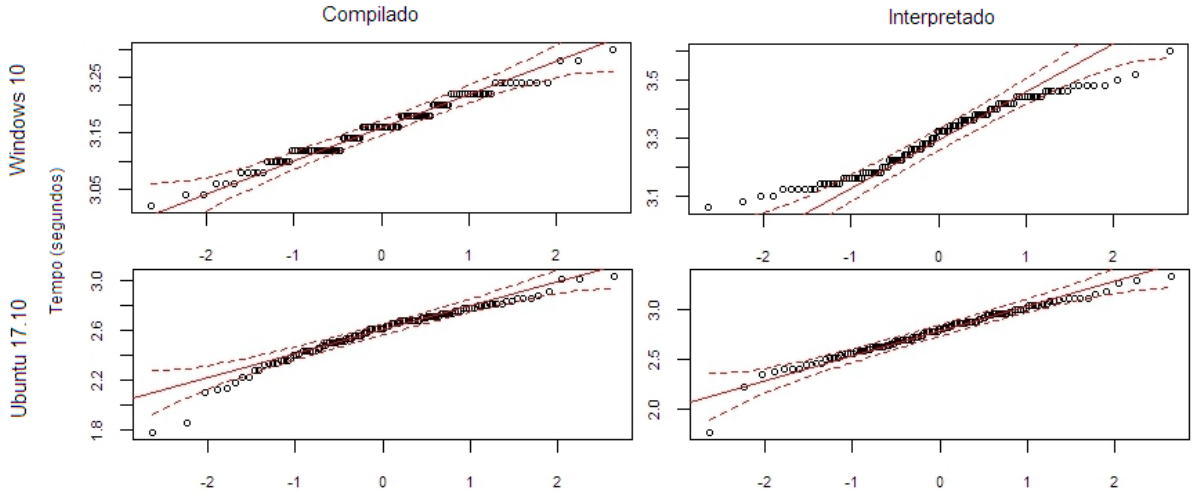

Figura 4 – Cruzamento dos resíduos das amostras finais vs. distribuição normal teórica.

Fonte: O AUTOR

A análise da normalidade dos resíduos na amostra definitiva foi feita com a aplicação do teste de Shapiro-Wilk com nível de significância de 0.05. Para todo o conjunto das 480 observações, o p-valor obtido indicou rejeição da hipótese de normalidade, sendo este $2.01x10^{-11}$. Neste sentido, o experimento foi alterado para uso de testes não paramétricos.

O cálculo do p-valor para detecção de que amostras provinham possivelmente de diferentes distribuições foi realizado com comparações por teste de Wilcoxon, nativo do R e

acessível via a função *pairwise.wilcox.test()*. O mesmo foi configurado com nível de significância de 0.01 e ajuste dos p-valores pelo método de Bonferroni. Os resultados obtidos são os exibidos na Listagem 2 onde, *cwin* trata-se de "Sistema Windows 10 com código R compilado", *cubu* diz respeito a "Sistema Ubuntu 17.10 com código R compilado", *iwin* é legenda para "Sistema Windows 10 com código R interpretado" e, por fim, *iubu* faz referência a "Sistema Ubuntu 17.10 com código R interpretado".

Como pode-se notar, o teste de Wilcoxon sugere que todas as amostras possuem distribuições diferentes entre si. Neste sentido, passou-se a aplicação do teste de Kruskal-Wallis para averiguação das magnitudes destas diferenças.

O teste de Kruskal-Wallis foi realizado por meio da função *kruskal()* do pacote *agricolae* de Mendiburu (2016). Este pacote contém diversas funcionalidades para realização de análise e planejamento de experimentos. É um pacote que possui testes específicos para área de agricultura, mas que pode ser utilizado também em outros contextos. Como no teste de Wilcoxon foram utilizados o ajuste pelo método de Bonferroni e nível de significância de 0.01. Os resultados calculados com Kruskal-Wallis podem ser visualizados na Listagem 3.

Listagem 2 – Aplicação de Teste de Wilcoxon

```
Pairwise comparisons using Wilcoxon rank sum test

     cwin    iwin    cubu
iwin < 2e-16 -       -
cubu < 2e-16 < 2e-16 -
iubu < 2e-16 < 2e-16 1.6e-09
```

Listagem 3 – Aplicação de Teste de Kruskal-Wallis

```
$parameters
  Df ntr  t.value alpha           test name.t
   3   4 3.162048  0.01 Kruskal-Wallis   base

$means
          rank    tempo        std   r  Min  Max
cubu  92.21667 2.592833 0.21380578 120 1.78 3.04
cwin 315.47083 3.160333 0.05413853 120 3.02 3.30
iubu 156.71250 2.788167 0.24011896 120 1.76 3.34
iwin 397.60000 3.301333 0.12232299 120 3.06 3.60

$comparison
            Difference pvalue sig.        LCL        UCL
cubu - cwin -223.25417      0  *** -250.30140 -196.20693
cubu - iubu  -64.49583      0  ***  -91.54307  -37.44860
cubu - iwin -305.38333      0  *** -332.43057 -278.33610
```

```
cwin - iubu  158.75833      0  ***  131.71110  185.80557
cwin - iwin  -82.12917      0  *** -109.17640  -55.08193
iubu - iwin -240.88750      0  *** -267.93473 -213.84027
```

Com base nos resultados numéricos pode-se perceber que, com um nível de confiança de 99% e, levando-se em conta os moldes estabelecidos para o protocolo de coleta, a melhor performance da linguagem R foi observada no cenário de código compilado e executado sob sistema operacional Ubuntu 17.10, visto que este cenário apresentou diferenças em vantagem frente a todos os outros concorrentes. Algumas outras observações podem ser feitas levando-se em conta a condução e os resultados deste experimento:

- Todos os cenários avaliados possuem evidências de diferenças entre si, sendo assim, conforme os testes, está demonstrado que a escolha de um sistema operacional (pelo menos entre Windows 10 e Ubuntu 17.10) pode influenciar diretamente no tempo de execução do código R produzido.
- Código R compilado apresentou melhor desempenho que código R interpretado em ambos os sistemas operacionais. Este fato mostra que, como já esperado, o ato de compilar um programa de computador pode acelerar a sua execução. Este fato reforça o uso de compilação como boa prática de programação.
- O código R, mesmo que interpretado em Ubuntu, ainda apresentou melhor desempenho que código compilado para Windows. Esta situação denota existir uma distância de qualidade entre os dois sistemas operacionais quando a finalidade do usuário for executar código produzido em linguagem R.
- A diferença entre código compilado e interpretado tanto em Windows, quanto em Ubuntu, mostrou-se pequena visto que as magnitudes das diferenças observadas estão, em ambos os casos, próximas entre si. Logo, quando se utilizando o mesmo sistema operacional, pode-se dizer que a compilação, apesar de útil, não é mais influente que a plataforma escolhida. Obviamente, levando-se em conta mais uma vez apenas Ubuntu 17.10 e Windows 10.
- Em consonância com as amostras iniciais, a maior diferença averiguada foi observada entre código interpretado em Windows para código compilado em Ubuntu, a favor do Ubuntu. Está maior diferença, de -305.38, quando observada em relação às médias em cada cenário (respectivamente 3.30 segundos em Windows interpretado e 2.59 segundos em Ubuntu compilado) mostra diferença de 0.71 segundos. Ubuntu compilado foi, portanto, 27.41% mais veloz. O pior caso em

Ubuntu (interpretado), frente ao melhor caso em Windows (compilado), neste mesmo tipo de análise, mostrou que Ubuntu foi 13.66% superior.

- A dispersão dos valores foi maior nos cenários com Ubuntu. Este fato pode indicar que este sistema modifica a forma de processamento de R a cada execução.

Conclusão

Com base nos resultados pode-se afirmar que, a um nível de confiança de 99%, as evidências conforme obtidas na condução deste experimento, apontam ser o sistema operacional Ubuntu 17.10 com código R compilado o cenário de melhor desempenho para a variável “tempo de execução” dada em segundos. Este resultado levou em conta ainda código interpretado em Windows 10, código interpretado em Ubuntu 17.10 e código compilado em Ubuntu 17.10.

O experimento foi conduzido com um código base específico com o qual buscou-se simular quatro tipos de rotinas comuns e que exigem esforço computacional. Este tipo de código nem sempre será similar ao produzido pelos programadores usuários de R, nesse sentido, recomenda-se para trabalhos futuros a realização do mesmo teste com códigos base diferentes e, também, sistemas operacionais diferentes.

Referências

MENDIBURU, Felipe de (2016). agricolae: Statistical Procedures for Agricultural Research. R package. Versão 1.2-4. https://CRAN.R-project.org/package=agricolae. Acesso em 5 de Abril de 2018.

MERSMANN, Olaf. 2014. **Microbenchmark Package**. R package. http://cran.r-project.org/web/packages/ microbenchmark/microbenchmark.pdf. Acesso em 20 de Março de 2018.

MILBORROW, Stephen. 2011. **JIT: Just-in-Time Compilation Package**. R package. http://www.milbo.users. sonic.net/ra/jit.html. Acesso em 20 de Março de 2018.

NAMETALA, Ciniro A.L. 2018. **Repositório de código fonte**. https://github.com/ciniro/r_performance. Acesso em 23 de Março de 2018.

R COMMUNITY. 2018. **Nabble R Forum**. R Foundation for Statistical Computing. http:// http://r.789695.n4.nabble.com. Acesso em 19 de Março de 2018.

ROSS, Noam. 2015. **FasteR! HigheR! StrongeR! - a Guide to Speeding up R Code for Busy People**. http://www.noamross.net/blog/2013/4/25/faster-talk.html. Acesso em 19 de Março de 2018.

SCOTT, Michael L. 2009. Programming Language Pragmatics. 3º edição. Elsevier.

TIERNEY, Luke. 2011. **Compiler Package**. R package. http://stat.ethz.ch/R-manual/R-devel/library/ compiler/html/compile.html. Acesso em 20 de Março de 2018.

UPADHYAY, Utkarsh. 2015. **Pre-Allocate Your Vectors**. http://musicallyut.blogspot.com.br/2012/07/ pre-allocate-your-vectors.html. Acesso em 22 de Março de 2018.

Anexos

Todos os códigos fonte em R produzidos para este experimento, bem como os *datasets* gerados, podem ser acessados publicamente no *link* disponível na referência (NAMETALA, 2018).

Agradecimentos

Este trabalho teve origem durante a disciplina de Análise e Planejamento de Experimentos do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, portanto, agradecemos ao Prof. Felipe Campelo França Pinto pela oportunidade e suporte.

Agradecemos também o apoio da NVIDIA *Corporation* com a doação de uma GPU Titan XP usada durante esta pesquisa.

Por fim, agradecemos ao Instituto Federal de Minas Gerais – Campus Bambuí pelo apoio financeiro e logístico.