Facultad de Ciencias

Departamento de Estadística

Carrera Ingeniería Estadística

PROYECTO DE TÍTULO II

Asignatura : Proyecto de Título II (220068)

Título : "Optimización del proceso de simulación para un test de bondad de ajuste usando las técnicas computacionales de paralelización disponibles en el lenguaje de programación R".

Realizado por : Katherine Deneken Guevara

Profesor Guía : Dr. Francisco Novoa Muñoz

Semestre : Segundo Semestre 2016

UNIVERSIDAD DEL BÍO-BÍO

www.ubiobio.cl

PROYECTO DE TÍTULO II

"Optimización del proceso de simulación para un test de bondad de ajuste usando las técnicas computacionales de paralelización disponibles en el lenguaje de programación R".

Katherine Deniss Deneken Guevara

Universidad del Bío-Bío

Departamento de Estadística

Diciembre 2016

Agradecimientos

Agradezco por sobre todo a Dios por cada oportunidad que en la vida me ha dado y jamás desampararme, dándome el privilegio de poder contar con mi familia, en especial mi amada Cofita y todos(as) mis amigos(as)(entre ellos, Any, Virgi, Andre, Jacita, Marce, Bayron, Ceci, Pam, Pauly y Zury) y mis seres queridos(as) que han sido un apoyo fundamental en el trascurso de mi carrera. También agradezco al Profesor Francisco Novoa, por la oportunidad dada, su paciencia y trato. Y en particular agradezco a nuestra querida Paty Toledo por todo el tiempo, cariño y dedicación que tiene para con nosotros, pues siento que alivianó inmensamente mi paso por la Universidad.

Resumen

Los datos de conteo bivariados surgen en varias disciplinas diferentes y la Distribución Poisson Bivariada (DPB), siendo una generalización de la Distribución de Poisson (DP), juega un rol importante al momento de modelarlos, siempre que dichos datos presenten una correlación no negativa. Un aspecto crucial de cualquier análisis de datos es contrastar la bondad de ajuste del modelo supuesto. Novoa-Muñoz & Jiménez-Gamero,2014 [33] propusieron tests de bondad de ajuste consistente para la DPB que para aproximar la distribución nula de los estadísticos involucrados emplea el método bootstrap paramétrico el cual requiere mucho tiempo de simulación. En este trabajo se experimenta el uso de técnicas de programación en paralelo mediante el lenguaje de programación R aplicados a la simulación del error tipo I del test $R_{n,w}(\hat{\theta_n})$ propuesto por los autores ya mencionados, utilizando el clúster Leftraru de NLHPC [19].

Palabras claves: Distribución Poisson Bivariada, programación paralela mediante R, bondad de ajuste, estimador boostrap, NLHPC.

Índice

Índice de Tablas

1. Introducción

La distribución Poisson (DP) ha sido ampliamente usada en teoría de distribuciones y es aplicada en muchas situaciones de la vida real, por ejemplo, es utilizada en: teoría de colas, aplicaciones biomédicas y epidemiológicas, industria, agricultura y ecología, accidentes, telefonía, comercio, geología, ingeniería, teoría de riesgos, seguros, demografía, etc. (ver por ejemplo, Maher (1982) [32], Karlis y Ntzoufras (2000) [25], (2003a) [26], (2003b) [27], (2005) [28], Rue y Salvesen (2000) [39]).

En la práctica, los datos de conteo bivariantes surgen en varias disciplinas diferentes y la distribución Poisson Bivariante (DPB), siendo una generalización de la DP, juega un rol importante al momento de modelarlos, siempre que dichos datos presenten una correlación no negativa.

Novoa-Muñoz y Jiménez-Gamero (2014) [33] (resumidos como N-J de aquí en adelante) propusieron tests de bondad de ajuste para la DPB que sí son consistentes.

Un test razonable para contrastar H_0 debería rechazar la hipótesis nula para valores grandes de cada test estadístico. Ahora, para determinar qué son los valores grandes en cada caso, se debe calcular la distribución nula de cada test estadístico o al menos una aproximación de cada una de ellas.

Puesto que las distribuciones nulas son desconocidas, y es muy alta la dificultad de estimarlas mediante la distribución asintótica nula pues depende del desconocido verdadero valor del vector de parámetros, en la práctica, se consideró otra forma de aproximar la distribución nula de los tests estadísticos, esto, con el método bootstrap paramétrico, que proporciona una solución útil al problema, aproximando consistentemente a la distribución nula de los estadísticos involucrados, (para una revisión detallada, ver N-J [33]).

Sin embargo al utilizar el método bootstrap paramétrico surge un nuevo inconveniente, el tiempo excesivo que tarda en dar resultados. Esto en términos económicos resulta ser muy caro, por el coste computacional que ello significa.

Esta debilidad que presenta el método bootstrap paramétrico (gran tiempo de

CPU) restringió la investigación de N-J, pues solamente se experimentó con una reducida cantidad de posibilidades de las infinitas que posee el vector de parámetros de la DPB. Además, sólo permite trabajar con tamaños de muestras limitados, esto trae como consecuencia desconocer el comportamiento del error tipo I para más tamaños muestrales, lo cual restringe también el estudio de la potencia del test.

En este proyecto de título II, se pretende agilizar el proceso, optimizando el tiempo de ejecución del primer test estadístico desarrollado por N-J [33], presentado en la ecuación (2) de la sección 2. Utilizando las técnicas de paralelización del lenguaje de programación R [36], estudiadas en el trascurso del proyecto de título I. Esto a fin probar si dichas técnicas permiten optimizar el tiempo de respuesta del programa o al menos logra alguna reducción de dicho tiempo, con intención de entregar una herramienta útil para cubrir la imperiosa necesidad de experimentar con más ternas del vector de parámetros y con nuevos tamaños muestrales.

Cabe destacar que en teoría [11] se señala que alguno de los métodos candidatos a poder ser paralelizados es el método bootstrap, pese a esto, también existen algunas limitaciones que podrían llegar a impedirlo, o a no lograr los resultados esperados con el programa en estudio, lo cual se muestra en la sección resultados.

“Powered@NLHPC: Esta investigación fue parcialmente apoyada por la infraestructura de supercómputo del NLHPC (ECM-02)”

2. Definiciones

2.1. Definición de la Distribución Poisson Bivariante

Se han dado varias definiciones para la DPB (ver por ejemplo, Kocherlakota y Kocherlakota (1992, pp. 87 − 90) [29], para una revisión detallada). En este trabajo, se considerará la siguiente (definida por Johnson, Kotz y Balakrishnan (1997, pp. 124 − 125) [21], que es la que ha recibido más atención en la literatura estadística (ver por ejemplo, Johnson, Kotz y Balakrishnan (1997)[21]).

Sean

$$X_1 = Y_1 + Y_3 \quad y \quad X_2 = Y_2 + Y_3,$$

donde Y_1, Y_2 e Y_3 son v.a. Poisson, mutuamente independientes con medias dadas por $\theta_1' = \theta_1 - \theta_3 > 0$, $\theta_2' = \theta_2 - \theta_3 > 0$ y $\theta_3 \geq 0$, respectivamente.

A la distribución conjunta del vector (X_1, X_2) se le denomina distribución **Poisson bivariante** (DPB) con parámetro $\theta = (\theta_1, \theta_2, \theta_3)$, lo cual se denota mediante $(X_1, X_2) \sim PB(\theta_1, \theta_2, \theta_3)$ o simplemente $(X_1, X_2) \sim PB(\theta)$.

La función de probabilidad conjunta de X_1 y X_2 está dada por

$$P_\theta(X_1 = x_1, X_2 = x_2) = exp(\theta_3 - \theta_1 - \theta_2) \sum_{i=0}^{min\{x_1,x_2\}} \frac{(\theta_1 - \theta_3)^{x_1-i}(\theta_2 - \theta_3)^{x_2-i}\theta_3^i}{(x_1 - i)!(x_2 - i)!i!},$$

donde $x_1, x_2 \in \mathbb{N}_0 = \{0, 1, 2, ...\}$

2.2. Bondad de Ajuste

Un aspecto crucial en cualquier análisis de datos es contrastar la bondad de ajuste de las observaciones con el modelo probabilístico supuesto, es decir, se contrasta si los datos provienen de la población que se supone.

Dadas las observaciones $X_1, X_2, ..., X_n$ independientes e idénticamente distribuidas, con distribución F, el objetivo es contrastar la hipótesis nula H_0 : "$F = F_0$". La hipótesis alternativa será H_1 : "$F \neq F_0$", F_0 puede ser una distribución totalmente especificada, o bien, especificada salvo por un número finito de parámetros N-J [33].

2.2.1. Test de bondad de ajuste en dimensión dos

Sean

$$\begin{aligned} &H_0 : (X_1, X_2) \sim PB(\theta_1, \theta_2, \theta_3), \text{ para algún } \theta_1, \theta_2, \theta_3 > 0, \\ &H_1 : (X_1, X_2) \text{ no se distribuye } PB(\theta_1, \theta_2, \theta_3), \forall \theta_1, \theta_2, \theta_3 > 0. \end{aligned} \tag{1}$$

Hasta donde es sabido, se pueden mencionar tres tests estadísticos para contrastar (1). El test estadístico T propuesto por Crockett (1979) [7], el test desarrollado por Loukas y Kemp (1986) [30] basado en lo que llamaron índice de dispersión bivariante y denotaron por I_B y el Test NI_B de Rayner y Best Rayner y Best (1995), que consiste en una modificación del test I_B recién mencionado. Lamentablemente estos tres tests no son consistentes, pues están basados en los momentos N-J [33].

2.3. Test estadístico $R_{n,w}(\hat{\theta}_n)$

Sean $X_1 = (X_{11}, X_{21}), X_2 = (X_{12}, X_{22}), ..., X_n = (X_{1n}, X_{2n})$ vectores aleatorios iid de $X = (X_1, X_2) \in \mathbb{N}_0^2$. Basándose en la muestra aleatoria $X_1, X_2, ..., X_n$, el objetivo es contrastar la hipótesis

$$H_0 : (X_1, X_2) \sim PB(\theta_1, \theta_2, \theta_3), \text{ para algún } \theta_1, \theta_2, \theta_3 \in \Theta,$$

contra la alternativa

$$H_1 : (X_1, X_2) \nsim PB(\theta_1, \theta_2, \theta_3), \forall \theta_1, \theta_2, \theta_3 \in \Theta,$$

donde $\Theta = \{\theta = (\theta_1, \theta_2, \theta_3) \in \mathbb{R}^3 : \theta_1 > \theta_3, \theta_2 > \theta_3, \theta_3 > 0\}$.

Con este propósito, Novoa-Muñoz y Jiménez-Gamero (2014) [33] aprovecharon algunas de las propiedades de la fgp que les permitieron proponer dos tests estadísticos. Siendo el primero de ellos el test $R_{n,w}(\hat{\theta}_n)$ que se presenta a continuación en la ecuación (2), con el cual se trabajará en este estudio dado que obtuvo una potencia lévemente superior que el segundo test estadístico propuesto por N-J [33].

$$R_{n,w}(\hat{\theta}) = n \int_0^1 \int_0^1 \left\{ g_n(u) - g(u; \hat{\theta}) \right\}^2 w(u) du \tag{2}$$

donde: n es el tamaño muestral,

$u = (u_1, u_2) \in [0,1]^2$,

$g(u) = E(u_1^X u_2^Y)$ es la llamada función generatriz de probabilidad,

$g_n(u) = \frac{1}{n}\sum_{i=1}^{n} u_1^{X_i} u_2^{Y_i}$ es la función generatriz de probabilidad empírica dada la muestra aleatoria $(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)$ del vector aleatorio $(X, Y) \sim PB(\theta)$.

Además, $g(u;\theta) = exp\{\theta_1(u_1 - 1) + \theta_2(u_2 - 1) + \theta_3(u_1 - 1)(u_2 - 1)\}$ es la función generatriz de probabilidad de la DPB y $\hat{\theta}$ es un estimador consistente del vector θ.

Por otra parte, la ecuación (2) utiliza la función de peso dada por

$$w(u) = u_1^{a_1} u_2^{a_2}, \qquad a_1, a_2 \in (-1, \infty),$$

la cual fue probada tan sólo para tres pares ordenados del vector (a_1, a_2), de los innumerables casos que existen. Es ideal poder experimentar más situaciones, pues ello impactaría en la potencia del test. Con lo cual sería posible encontrar una mejor potencia del test frente a una alternativa particular.

3. Instrumento

Para llevar a cabo este Proyecto de Título fue necesario e imprescindible contar con algún instrumento de programación para obtener resultados y conclusiones sobre la posibilidad de paralelizar el test de bondad de ajuste $R_{n,w}(\hat{\theta}_n)$ [33] en estudio.

R [36]con su lenguaje de programación en conjunto con la posibilidad otorgada de haber tenido acceso al cluster Leftraru del Laboratorio Nacional de Computación de Alto Rendimiento (NLHPC) [19] son el instrumento.

La finalidad de utilizarlo es reducir en lo posible los tiempos de ejecución del código que se describe en la sección 6.1, procurando a la vez conservar la efectividad de sus respuestas al aplicar los métodos de paralelización disponibles en R.

Para implementar el uso de dicho instrumento se requieren softwares para ejecutar tareas en el cluster Leftraru desde un computador externo al sistema del NLHPC [19]. Se puede acceder al cluster Leftraru a través del protocolo SSH [18] a leftraru.nlhpc.cl en el puerto 22, con el nombre de usuario y clave otorgados por NLHPC [19]. Esto, utilizando WinSCP [16] que a su vez también por medio del protocolo SSH permite utilizar PuTTY [20].

Los softwares recién mencionados se describen en términos simples a continuación:

-**Secure SHell** (cuya traducción sería intérprete de comandos seguro) es un protocolo de comunicación para controlar un ordenador en remoto a través de una CLI (Command Line Interface -Interfaz de Línea de Comandos- también llamada: "shell"). Sirve para conectarse con un computador ante el cual no se está físicamente, bien porque está en una sala de servidores refrigerada, porque no tiene teclado ni pantalla, entre otros. SSH es un protocolo seguro, la información viaja codificada, lo cual es fundamental a la hora de conectarse a través de Internet.

-**WinSCP** es una herramienta que se encarga de consumar las operaciones primordiales de los archivos; tal como lo son las descargas y subidas de estos. Asimismo es posible a través de él cambiar los nombres de directorios y archivos, crear enlaces

simbólicos y accesos directos, crear nuevos directorios, cambiar las propiedades de archivos y carpetas actuales. Es práctico de usar para los individuos que no utilizan de manera común la Consola de algún terminal Linux, desde Windows.

-**PuTTY** Es un programa cliente para SSH y telnet, desarrollado originalmente por Simon Tatham para la plataforma Windows. PuTTY es un software de código abierto que está disponible con el código fuente y se desarrolla con el apoyo de un grupo de voluntarios. Estos protocolos son utilizados para ejecutar una sesión remota en un equipo, a través de una red. PuTTY implementa el cliente final de ese período de sesiones: el extremo en el que se visualiza la sesión, en lugar del extremo en el que se ejecuta.

En términos muy simples: se ejecuta PuTTy en una máquina Windows, abriendose una ventana por la cual se conecta a (por ejemplo) una máquina Unix. Entonces, todo lo que se teclea en la ventana se envía directamente a la máquina Unix, y todo lo que la máquina Unix envía de vuelta se muestra en la ventana.

PuTTY no procesa los comandos que escriba en él. No es más que una herramienta de comunicación. Por lo tanto, la gama exacta de comandos que puede utilizar no dependerá de PuTTy, pero sí de qué tipo de equipo que se haya conectado y qué software se está ejecutando en él.

4. Metodología

R cuenta con una gran variedad de técnicas de paralelización mediante su propio lenguaje de programación, algunas de estas técnicas fueron estudiadas en el trasncurso del proyecto de título I, de las cuales se concluyó que el uso de la librería "foreach" era lo recomendable pues se consideraba una técnica que gobernaba al resto, con la finalidad de reducir el tiempo de respuesta del programa en estudio.

Basado en lo anterior el siguiente paso fue conseguir una cuenta de acceso al cluster Leftraru, solicitada a través de la página de NLHPC [19].

Una vez aceptada la solicitud de la cuenta de acceso al cluster mencionado, lo siguiente fue comprender el funcionamiento y requisitos de dicha plataforma para poder acceder y trabajar mediante ella, realizando simulaciones del primer test de bondad de ajuste con los diferentes tipos de paralelización estudiados e ir modificando sus variantes hasta determinar cuál es la mejor opción.

Para el procesamiento de los datos se utilizó el software estadístico R ya mencionado en sus versiones R/3.1.2 y R/3.2.0. disponibles en el cluster leftraru utilizado. Mientras que para la edición del texto se utilizó el procesador de texto avanzado LáTeX.

5. Objetivos

5.1. Objetivo General

Optimizar el tiempo de simulación del error tipo I del primer test propuesto por N-J [33], utilizando el lenguaje de programación R en el clúster Leftraru.

5.2. Objetivos Específicos

1. Obtener una cuenta en los sistemas del NLHPC [19].

2. Comprender el programa a paralelizar.

3. Programar en R a través de la plataforma de súper computadores [19].

4. Aplicar técnicas de paralelización del lenguaje R [35] al programa en cuestión.

5. Comparar resultados.

6. Resultados

Las propiedades asintóticas estudiadas en N-J [33], describen el comportamiento de los tests propuestos para muestras de tamaño grande. Para comparar el tiempo de ejecución y verificar la potencia del test propuesto para este estudio, en modo secuencial como en paralelo, se ha llevado a cabo un experimento de simulación utilizando el código computacional del test $R_{n,w}(\hat{\theta})$ diseñado para estudiar la bondad de la aproximación bootstrap para una muestra de tamaño finito realizado por N-J [33].

En esta sección se describe dicho experimento y se da un resumen de los resultados obtenidos.

Todos los cálculos computacionales realizados en este trabajo se llevaron a cabo mediante el uso de programas escritos en el lenguaje R [35], los cuales fueron lanzados desde PuTTy [20] por medio de WinSCP [16] que conectándose mediante el protocolo SSH [18] a leftraru.nlhpc.cl en el puerto 22 permiten la conexión del computador con el cluster Leftraru, mediante el uso de la cuenta otorgada por NLHPC [19].

Para calcular $R_{n,w}(\hat{\theta_n})$ es necesario dar una forma explícita a la función de peso w. En el caso univariante, Baringhaus et al. (2000) [1] consideraron la función de peso $w(u) = u^a$, con $a \geq 0$. Varias extensiones son posibles. Aquí tal como en N-J [33] se ha tenido en cuenta la siguiente

$$w(u; a_1, a_2) = u_1^{a_1} u_2^{a_2}. \tag{3}$$

Observar que las únicas restricciones que se impusieron a la función de peso son que w sea positiva casi en todas partes en $[0,1]^2$ y que $\int_0^1 \int_0^1 w(u)du < \infty$. La función $w(u; a_1, a_2)$ dada en (3) cumple dichas condiciones siempre que $a_i > -1, i = 1, 2$.

6.0.1. Técnicas de paralelización utilizadas

De los diferentes métodos disponibles en el lenguaje de programación R, se optó por utilizar las que se especifican a continuación:

Una aproximación común a la paralelización es ver si las iteraciones dentro de un bucle pueden ser realizadas independientemente, y si es así, intentar ejecutar las iteraciones simultáneamente en lugar de secuencialmente.

Los paquetes "foreach"[12] e "iterators" [14] creados por la revolución informática [37] proporcionan un marco práctico para la computación paralela en R. Con estos dos paquetes, sólo es necesario escribir una versión del código para todos los backends paralelos, que son una interfaz que proporciona un mecanismo necesario para ejecutar los bucles foreach en paralelo. Lo cual se conoce como "un anillo para gobernarlos a todos".

El paquete Foreach ofrece una nueva construcción de bucle para ejecutar código R repetidamente. Con la desconcertante variedad de construcciones de bucle existentes, es posible dudar de que aun exista una necesidad de otra construcción. La razón para usar dicho paquete es que soporta ejecución en paralelo, es decir, se pueden ejecutar operaciones repetidas en múltiples procesadores/núcleos computacionales, o en varios nodos de un conglomerado, para lo cual se debe tener un backend registrado.

Foreach es un conjunto de herramientas que le permiten ejecutar prácticamente cualquier cosa que pueda ser expresado como un bucle for como un conjunto de tareas paralelas.

La comparación de la salida foreach con la de un bucle similar muestra una diferencia, foreach devuelve una lista que contiene el valor devuelto por cada cálculo. Un bucle for, por contraste, sólo devuelve el valor de su último cálculo y se basa en efectos secundarios definidos por el usuario para hacer su trabajo.

Los operadores foreach $\%do\%$ y $\%dopar\%$ se utilizan entre el foreach y el cuerpo de bucle, proporcionan una estructura de bucle que se puede ver como un híbrido del estándar para la función bucle y lapply. Se parece al bucle for, y evalúan una expresión, en lugar de una función (como en lapply), pero su propósito es devolver un valor (una lista, por defecto), en lugar de causar efectos secundarios. Esto facilita la paralelización, pero parece más natural para las personas que prefieren los bucles

a lapply [37].

Cuando se utiliza foreach, no se crea una matriz para asignar valores a ella. En su lugar, el Bucle interno, devuelve las columnas de la matriz de resultados como vectores, que se combinan en el bucle exterior, el código es ligeramente más limpio y tiene la ventaja de ser paralelizado.

.combine es una función que se utiliza para procesar los resultados de las tareas a medida que se generan. Esto se puede especificar como una función o una cadena de caracteres no vacía que nombra la función. Especificar 'cés útil para concatenar los resultados en un vector, los valores 'cbind'y 'rbind'pueden combinar vectores en una matriz columna y fila, respectivamente. Los carácteres '+'y '∗'pueden usarse para procesar datos numéricos. Por defecto, los resultados se devuelven en una lista.

.multicombine indica si la función .combine puede aceptar más de dos argumentos. Si se especifica una función .combine arbitraria, de forma predeterminada, esa función siempre se llamará con dos argumentos. Si puede tomar más que dos argumentos, entonces el ajuste .multicombine a TRUE podría mejorar el desempeño. El valor predeterminado es FALSE a menos que la función .combine sea cbind, rbind o c, que se sabe que toman más de dos argumentos.

Se puede paralelizar la operación inmediatamente reemplazando %*do*% con %*dopar*%, sin embargo se requiere de un backend paralelo registrado previamente. De lo contrario entregará la siguiente advertencia:

"Ejecutando %*dopar*% secuencialmente: no hay backend paralelo registrado".

Los detalles de registrar los backends paralelos difieren, dado que cada uno considera sus propios argumentos específicos.

Una característica importante de foreach es el operador de anidación % : %. Al igual que los operadores %*do*% y %*dopar*% es un operador binario, utilizado para crear bucles foreach anidados, opera en dos objetos foreach. También devuelve un objeto foreach, que es esencialmente una fusión especial de sus operandos (ver

Vignette ("anidado") en el indicador R para obtener más detalles).

Cuando se realizan paralelizaciones de bucles anidados, siempre hay una cuestión de qué bucle paralelizar. El consejo estándar es paralelizar el bucle exterior. Esto resulta en tareas individuales más grandes, y las tareas más grandes a menudo se pueden realizar más eficientemente que las tareas más pequeñas.

Sin embargo, si el bucle no tiene muchas iteraciones y las tareas ya son grandes, paralelizando el bucle externo resulta en un pequeño número de tareas enormes, que pueden no permitirle utilizar todos sus procesadores y también puede resultar en problemas de equilibrio de carga. También se podría paralelizar un bucle interno en su lugar, pero podría ser ineficiente porque se está esperando repetidamente que todos los resultados sean devueltos cada vez a través del bucle exterior. Y si las tareas y el número de iteraciones varían en tamaño, entonces es realmente difícil saber qué bucle paralelizar. Y por ahora sólo queda probar.

Lo ideal sería que todas las tareas fuera completamente independientes entre sí, para que todos puedan ejecutarse en paralelo. Pensar en los bucles como especificando un único flujo de tareas. Sólo se tiene que tener cuidado de procesar todos los resultados correctamente, dependiendo de la iteración del lazo interno de donde provenían y es exactamente lo que hace el operador $\%:\%$, convierte varios bucles foreach en un único lazo.

Por supuesto, en realidad sólo se pueden ejecutar tantas tareas en paralelo como procesadores se tiene, pero el backend paralelo se encarga de todo eso, el argumento .combine de foreach permite especificar cómo deben procesarse los resultados(para más información sobre las llamadas anidadas de foreach, vea la vignette "Nesting foreach Bucles" en el paquete foreach).

Una cosa a tener en cuenta son las posibles asignaciones a las variables en los bucles. Pues al hacerlo con foreach, para que el código funcione correctamente en paralelo, se necesita tener cuidado de no utilizar asignaciones de variables como una forma de comunicarse entre diferentes iteraciones del bucle. Eso es lo que los programadores paralelos llaman una "dependencia de bucle". También es importante

tener en cuenta escribir dentro del bucle las variables que se definieron antes de él. En muchos sistemas de computación paralela, se tienen que exportar explícitamente las variables, de diferentes maneras.

Cabe señalar que es importante ser muy cuidadoso de hacer uso de vectores en bucles. La regla primaria para obtener un buen desempeño en R es usar operaciones vectoriales cuando sea posible.

La idea general es utilizar operaciones vectoriales para los cálculos e ir utilizando foreach para ejecutar las operaciones vectoriales en paralelo.

Del paquete Iterators, un iterador es un tipo especial de objeto que generaliza la noción de una variable de bucle. Cuando se ha pasado como argumento a una función que sabe qué hacer con ella, el Iterators suministra una secuencia de los valores. También mantiene información sobre su estado, en particular su índice actual.

Incluye una serie de funciones para crear iteraciones, el más simple del cual es iter, que toma prácticamente cualquier objeto R y lo convierte en un objeto Iterador.

Por ejemplo, la función foreach toma iterables como argumentos. Llama al iter sobre estos argumentos para crear iteradores para ellos. Mediante la definición del método iter para todos los iteradores se pueden pasar al foreach creado, usando cualquier método que se elija. Por lo tanto, se pueden pasar vectores, listas o Iteradores a foreach, y todos ellos son procesados por foreach exactamente de la misma manera.

El problema con foreach es que gasta mucho tiempo en la comunicación entre los núcleos para obtener los resultados de los diferentes núcleos y hacer que estos encajen. Foreach es sólo recomendable si se tienen relativamente pocas rondas a través de funciones que llevan mucho tiempo.

Actualmente, Revolution R Enterprise, incluye el paquete doParallel, que es un "backend paralelo"para el paquete foreach. Proporciona un mecanismo necesario para ejecutar los bucles foreach en paralelo.

doParallel [50] actúa como una interfaz entre foreach y el paquete Parallel de R 2.14.0 y posteriores. Es esencialmente una fusión del paquete Multicore, que fue

escrito por Simon Urbanek, y el paquete de SNOW, que fue escrito por Luke Tierney y otros. La funcionalidad de varios núcleos admite varios trabajadores sólo en los sistemas operativos que soportan la llamada al sistema de la FORK; esto excluye Windows. De forma predeterminada, doParallel utiliza funciones multicore en sistemas Unix-like y funcionalidad de SNOW en Windows. Multicore sólo ejecuta tareas en un solo equipo, no un grupo. Sin embargo, se puede utilizar la funcionalidad de SNOW para ejecutar en un clúster, utilizando sistemas operativos similares a Unix, Windows o incluso una combinación. No tiene sentido utilizar doParallel y paralell en una máquina con sólo un procesador y un solo núcleo. Para obtener una mejora de velocidad, debe ejecutarse en una máquina con múltiples procesadores, múltiples núcleos o ambos.

Para registrar doParallel para ser utilizado con foreach, debe llamar a la función registerDoParallel. Si llama a esto sin argumentos, en Windows obtendrá tres trabajadores y en sistemas similares a Unix obtendrá un número de trabajadores igual a aproximadamente la mitad del número de núcleos en su sistema.

También puede especificar un clúster (creado por la función makeCluster) o varios núcleos. Los cores indican el número de procesos de trabajo que doParallel utilizará para ejecutar tareas, que por defecto será igual a la mitad del número total de núcleos en la máquina. Sin embargo, es necesario especificar un valor para ello. Por defecto, doParallel utilizará el valor de los "núcleos", tal como se especifica con la función "*options*" estándar. Si no está configurado, entonces doParallel intentará detectar el número de núcleos, y utilizar la mitad de trabajadores.

Como fue mencionado, Parallel funciona para un solo computador por lo cual no requiere de MPI y tiene funcionalidad similar a SNOW utilizando conexiones de socket o funcionalidad similar a Multicore usando bifurcación (Sólo en Linux). Para ello requiere la instalación de dichos paquetes, los cuales se describen brevemente a continuación:

El paquete doMC es un "backend paralelo"para el paquete foreach, que en conjunto depende del paquete Multicore. Proporciona un mecanismo necesario para

ejecutar bucles foreach en paralelo.

La funcionalidad multicore actualmente sólo funciona con sistemas operativos tipo Fork. Además, el Multicore sólo ejecuta tareas en un solo computador, no un grupo. Esto significa que es inútil usar doMC y Multicore en una máquina con un solo procesador con un único núcleo.

Se debe tener precaución debido a que la funcionalidad Multicore inicia a sus trabajadores utilizando FORK, tiene algunas limitaciones. Algunas operaciones no se pueden realizar correctamente mediante procesos bifurcados. Por ejemplo, los objetos de conexión muy probablemente no funcionarán. En algunos casos, esto podría hacer que un objeto se dañen y la sesión R se bloquee, Además, normalmente no es seguro ejecutar doMC y multicore desde un entorno GUI [2].

Lamentablemente en el caso aplicado al programa en estudio, utilizando doMC, no se pudo llevar a cabo un experimento concreto, pues en el clúster no se pudo instalar la librería Multicore requerida, por problemas de actualización. En los intentos de paralelización realizados con doMC(que si se pudo instalar) sólo corrió en modo secuencial, lo cual no produjo reducciones de tiempo. Por lo anterior, experimentos con el backen multicore tampoco fueron posibles.

Respecto al paquete doSNOW funciona similar al SNOW ("Simple Network of Workstations"), [15] como su nombre lo indica, con este paquete, se puede construir una red sencilla de estaciones de trabajo sobre la marcha. Permite usar un cluster tradicional para realizar tareas en paralelo, es decir, permite aprovechar al máximo varios computadores Linux en un laboratorio sin el manejo directo de cada uno de ellos por individual. Es probablemente el paquete de programación en paralelo más popular disponible para R. Escrito por Luke Tierney, AJ Rossini, Na Li, y H. Ševcíková, y se mantiene activa por Luke Tierney. Es un paquete consistente, lanzado por primera vez en el "Integral R Archivo de Red"(CRAN) en 2003.

Este método se considera bueno para su uso en grupos tradicionales, en especial si se encuentra activo MPI ya que aprovecha su velocidad de forma menos compleja

sin embargo es difícil de configurar. Es portátil y corre en Linux, Mac OS X y Windows.

Snow usa una arquitectura maestro/esclavo, donde el maestro envía trabajos a los esclavos y luego de realizado, los esclavos devuelven los resultados al maestro. Puede usar diferentes medios de transporte para comunicar el maestro con los esclavos, lo cual lo hace altamente portable. Más especificamente, es posible utilizar sockets, MPI (message passage interface), PVM (parallel virtual machines) o NetWorkSpaces.

Para poder usar SNOW hay que saber cuántos núcleos tiene cada computador para así asignarlos al trabajo y haber instalado previamente tanto R como el paquete SNOW en cada una de las máquinas. Sin embargo, La mejora de velocidad no siempre es espectacular, y hay al menos dos razones importantes para esto; en primer lugar, que las computadoras que se añadan no sean tan poderosos como el computador de origen; en segundo lugar, la sobrecarga de comunicación entre computadores es mucho más pesada que la de uno solo.

Por lo tanto, el aumento en el número de máquinas o núcleos, no significa automáticamente un gran aumento de velocidad, porque hay que darse cuenta de los gastos generales. Así que se debe ser cuidadoso y hacer los cálculos cada vez que se piense en la ampliación de un grupo.

Por las limitaciones de tiempo que puede producir una mala implementación de SNOW y doSNOW junto a la dificultad extra de configurar correctamente MPI, se optó por utilizar, una versión más práctica del mismo, esto es la ejecución del paquete doParallel mediante el estilo de doSNOW, lo cual se denomina DoParallelSNOW.

A fin de encontrar mejoras se utilizaron varias técnicas de programación paralela, sin embargo de todas, DoParallelSNOW fue la que presentó mejores resultados, lo primero fue una revisión del programa del test mencionado, a fin de detectar los puntos que producían mayor retardo en responder y se determinó que ese retardo ocurre en el método bootstrap para calcular el estadístico $R_{n,w}(\hat{\theta}_n)$, con las características mencionadas en la sección 6.1, producto de la necesidad de iterar mil veces la simulación mediante tres ciclos for.

Entre las técnicas utilizadas se intentó cambiar ciclos for del programa en estudio por funciones de Lapply, sin embargo los resultados obtenidos no fueron los esperados puesto que las mejoras que presentó en tiempo, dicho cambio, lo limitó el hecho de no arrojar las respuestas solicitadas. Dicha limitación ocurrió también al utilizar otras técnicas de las mencionadas y fué dificil de solucionar dado la escasa literatura que se refiere al tema.

Por lo cual se continuó la aplicación DoParallelSNOW de la siquiente forma:

- Primero se instalaron de las librerías: methods, Runuran, iterators, foreach, parallel y doParallel. Cada una instalada previamente en el clúster leftraru.

- Luego se construyó y registró el cluster a utilizar mediante doParallel, como se muestra:

  $cl - makeCluster(4, type =" FORK", outfile =" ETI - paralelo.txt")$

  $registerDoParallel(cl)$

  donde, cl es la variable que guarda el cluster generado, 4 es la cantidad de trabajadores utilizados; FORK es el sistema que permite dividir en ramas e ir por caminos separados y además contiene automáticamente las variables de entorno; outfile, es el comando que permite exportar las respuestas del programa a un archivo con la codificación que se requiera, en este caso ETI-paralelo.txt. Finalmente registerDoParallel() es el comando que permite el registro del cluster construido, con las características mencionadas.

  Algo importante ante la creación de un clúster es no olvidar cerrarlo, esto mediante el comando $stopCluster(cl)$.

- Sin duda una de las partes fundamentales es poder determinar el tiempo que tarda cada simulación, para lo cual se utilizó el compando $system.time$.

- Con varios intentos diferentes de convertir en foreach los ciclos for que se consideraron los más lentos en ejecutar, los que al ser llamados podrían haber

retardado el proceso, todos, alguno, algunos y ninguno. Lo que dio mejor resultado fue lo que se muestra en la tabla 2, mientras que la tabla número 1, muestra el extracto del programa en su versión original.

Tabla 1: Extracto del test $R_{n,a}$ original.

```
# a_12 es la opcion del par (a1,a2)
for(a_12 in par:par)
{
 # 1. Se calculan los valores observados para la muestra
 # original
 # r_obs es el valor observado del estadistico r_n,a
 # a1 y a2 son los valores de a1 y a2 en IN_0 de la
 # funcion de peso
 aa <- paste(('',a1[a_12],'','',a2[a_12],'')'',sep='''')
 vp_r <-  rep(0  , M)
 for( m in 1:M )
 {
  XY <- rbind(X[m,],Y[m,])
  theta_estim <- EstimadorMV(XY)
  r_obs <-  R(n,XY,theta_estim[1],theta_estim[2],
  theta_estim[3],a1[a_12],a2[a_12])
  r_boot <- rep(0,B)
  for (b in 1:B)
  {
   sigue <- ''no''
   while(sigue==''no'')
   {
    #2. Se generan B muestras bootstraps Mboot desde la
    #DPB PB(theta_estim)
```

Nota: La tabla 1 continúa en la siguiente página.

```
    X_PBboot <- gmpb(n,theta_estim[1],theta_estim[2],
    theta_estim[3])
    #3. Se calcula el estimador bootstrap de theta
    theta_est_boot <- EstimadorMV(X_PBboot)
    if(theta_est_boot[1]>theta_est_boot[3] && theta_est_
    boot[2]>theta_est_boot[3] && theta_est_boot[3]>0)
    sigue <- ''si''
   }
   #4. Se evaluan los estadisticos en cada muestra bootstrap
   r_boot[b] <- R(n,X_PBboot,theta_est_boot[1],
   theta_est_boot[2],theta_est_boot[3],a1[a_12],a2[a_12])
  }
  #5. Se acumula una aproximacion del valor-p para cada
  #estadistico
  ind_r <- rep(0,B)
  ind_r[r_boot >= r_obs] <- 1
  vp_r[m] <- sum(ind_r)/B
 }
 ind2_r <- ind3_r <- rep(0,M)
 ind2_r[vp_r <= .05] <- 1
 error05_r <- sum(ind2_r)/M
 ind3_r[vp_r <= .1] <- 1
 error10_r <- sum(ind3_r)/M
 A <- ks.test(vp_r,''punif'')
 a <- round(A[[2]],6)
 if(a==0) a <- ''< 2.2e-16''
}
```

Fuente: Elaboración propia.

Tabla 2: Extracto del test $R_{n,a}$ paralelizado con foreach y el backend DoParallelSNOW registrado, corriendo con 4 workers.

```
# a_12 es la opcion del par (a1,a2)
foreach(a_12=par:par, .multicombine=TRUE,.combine=
'rbind')%dopar%
{
 # 1. Se calculan los valores observados para la muestra
 # original
 # r_obs es el valor observado del estadistico r_n,a
 # a1 y a2 son los valores de a1 y a2 en IN_0 de la
 funcion de peso
 aa <- paste(('',a1[a_12],'','',a2[a_12],'')'',sep='''')
 vp_r <- rep(0 , M)
 foreach ( m=1:M )%dopar%
 {
  XY <- rbind(X[m,],Y[m,])
  theta_estim <- EstimadorMV(XY)
  r_obs <- R(n,XY,theta_estim[1],theta_estim[2],
  theta_estim[3],a1[a_12],a2[a_12])
  r_boot <- rep(0,B)
  foreach (b=1:B,.multicombine=TRUE,
  .combine= 'rbind')%dopar%
  {
   sigue <- ''no''
   while(sigue==''no'')
   {
    #2. Se generan B muestras bootstraps Mboot desde la
    #DPB PB(theta_estim)
```

Nota: La tabla 2 continúa en la siguiente página.

```
    X_PBboot <- gmpb(n,theta_estim[1],theta_estim[2],
    theta_estim[3])
    #3. Se calcula el estimador bootstrap de theta
    theta_est_boot <- EstimadorMV(X_PBboot)
    if(theta_est_boot[1]>theta_est_boot[3] &&theta_est_
    boot[2]>theta_est_boot[3] && theta_est_boot[3]>0)
    sigue <- ''si''
   }
   #4. Se evaluan los estadisticos en cada muestra bootstrap
   r_boot[b] <- R(n,X_PBboot,theta_est_boot[1],
   theta_est_boot[2],theta_est_boot[3],a1[a_12],a2[a_12])
  }
  #5. Se acumula una aproximacion del valor-p para cada
  #estadistico
  ind_r <- rep(0,B)
  ind_r[r_boot >= r_obs] <- 1
  vp_r[m] <- sum(ind_r)/B
 }
 ind2_r <- ind3_r <- rep(0,M)
 ind2_r[vp_r <= .05] <- 1
 error05_r <- sum(ind2_r)/M
 ind3_r[vp_r <= .1] <- 1
 error10_r <- sum(ind3_r)/M
 A <- ks.test(vp_r,''punif'')
 a <- round(A[[2]],6)
 if(a==0) a <- ''< 2.2e-16''
}
```

Fuente: Elaboración propia.

Para obtener información sobre el backend Parallel se pueden utilizar:

- $getDoParWorkers()$, permite saber cuántos trabajadores foreach se van a utilizar. Si no se ha registrado Un backend paralelo, o si su máquina sólo tiene un núcleo, getDoParWorkers devolverá 1.
- $getDoParName()$, entrega el nombre del backend actualmente registrado.
- $getDoParVersion()$, se obtiene la versión del backend actualmente registrado.

Cabe mencionar que el paquete doMPI es un "backend paralelo"para el paquete foreach. No es un sistema de programación paralelo, sino un marco de programación paralelo, un sistema de programación paralela para hacer el trabajo realmente en paralelo. El paquete doMPI actúa como un adaptador al paquete Rmpi, que a su vez es una interfaz R para una implementación de MPI [48].

MPI, O Message Passing Interface, es una especificación para una API para pasar mensajes entre diferentes computadores. Hay una serie de implementaciones MPI disponibles que permiten que los datos sean movidos entre computadores de manera bastante eficiente [46].

La programación con MPI es bastante difícil, sin embargo, ya que es una API bastante grande y compleja. Por ejemplo, el paquete Rmpi define unas 110 funciones R, sólo algunas de ellas son utilizadas. Y el estándar MPI incluye muchas más funciones que no son compatibles con Rmpi, como las funciones del archivo. Por supuesto, es necesario aprender tansolo un pequeño porcentaje de esas funciones para empezar a usar MPI eficazmente, pero puede tomar un tiempo sólo para averiguar qué funciones son realmente importante, y cuáles puede ignorar con seguridad [46].

Desafortunadamente, todavía hay una gran variedad de problemas que se pueden encontrar. Primero, se necesita tener MPI instalado en los computadores, y entonces se tiene que construir e instalar Rmpi para usar esa Instalación MPI. Ahí es donde muchas personas se encuentran con problemas, especialmente en Windows. Sin em-

bargo, En Debian / Ubuntu, es más sencillo instalar Open MPI, y Rmpi funciona bastante bien por lo general [46].

Otra cosa a tener en cuenta es que se tienen que ejecutar los programas R de manera diferente a fin de utilizar varios equipos, y el método exacto varía en función de cada implementación MPI. Si se acaba de iniciar una sesión normal de R, sus trabajadores (wokers) sólo se ejecutarán en una máquina local. Eso es práctico para probarlo, pero la finalidad de usar doMPI es ejecutarlo en varios computadores. Para hacer eso se necesita iniciar una sesión R usando un comando como mpirun o mpiexec, dependiendo del MPI instalado [46].

Existen documentos que ayudan a empezar, pero directamente no hay tutoriales explícitamente detallados de cómo implementarlo, por lo que se recomienda unirse a la lista de correo R-sig-hpc [46].

Una vez instalado R y MPI en sus equipos, tendrá que instalar DoMPI y los paquetes de los que depende. En este caso eso se realizó solicitando dichas librerías al equipo de soporte NLHPC.

Una vez que haya iniciado una sesión R e instalado Rmpi, se debe cargar el paquete doMPI. Sin duda se recibirá un error en este momento si Rmpi no está instalado correctamente. Si eso sucede, se debe reiniciar la sesión R, cargar Rmpi, hacer una copia del mensaje de error, averiguar acerca del error, y de no encontrar una solución, redactar una petición de ayuda a Sig-hpc incluyendo todo el mensaje de error y posiblemente, alguien en la lista que haya tenido ese error antes, ofrecerá consejos sobre cómo solucionarlo. Una vez que se puede cargar con éxito doMPI, el siguiente paso es crear un objeto de clúster MPI. Existen varias opciones, pero aquí hay una manera de hacerlo:

$Cl < -startMPIcluster(count = 2)$

Esto inicia dos workers de clúster, o esclavos, como MPI los llama. Hay que tener en cuenta que si utiliza Open MPI, esos dos procesos empezarán inmediatamente a usar la mayor cantidad de CPU posible. Esto es un problema conocido con Open MPI.

Están esperando que el proceso maestro les envíe un mensaje, pero están "ocupados esperando"por ese mensaje. Eso no tiene mucho sentido en este contexto, pero puede mejorar el rendimiento en otros. En cualquier caso, es un problema que el grupo Open MPI podría abordar en un futuro lanzamiento [46]. El siguiente paso es registrar el objeto de clúster MPI con el paquete foreach. Esto se hace con la función "registerDoMPI":

$RegisterDoMPI(cl)$

Esto le dice a foreach que desea usar doMPI como el backend paralelo, y que desea utilizar el objeto de clúster MPI especificado para ejecutar las tareas. Si no se registra un objeto de clúster, las tareas no se ejecutarán en paralelo, aunque se utilice foreach con el operador $\%dopar\%$.

Cuando haya terminado de usar el objeto de clúster MPI, es importante apagarlo, de lo contrario se pueden perder los procesos en el clúster. Eso con la función closeCluster:

$CloseCluster(cl)$

Finalmente, es probable que también se deba llamar a mpi.quit para salir de la sesión R al usar doMPI.

Para que todo funcione correctamente, todos los puestos de trabajo deben tener la misma cuenta de usuario, con el mismo directorio de inicio, con ssh sin contraseña Habilitado y todo el mismo software instalado utilizando las mismas rutas de acceso de archivos. Por ejemplo, R debe ser Instalado en el mismo lugar en todas las máquinas, y el paquete doMPI debe instalarse en todas las máquinas, también. Es un requisito bastante grande, pero esa es la forma en que la mayoría de la gente haces cosas con MPI, y así es como se configuran muchos clústeres de computadores (Para más información, ver PackagedoMPI [48]).

6.1. Datos Simulados

Del código escrito en lenguaje R, hecho para estudiar la bondad de la aproximación bootstrap en muestras de tamaño finito para varios tests propuestos por N-J [33], se analizó para este estudio sólo el primer test $R_{n,w}(\hat{\theta})$ N-J [33]. el cual funciona como se describe a continuación.

Generada una muestra de tamaño $n = 30$ de la distribución $PB(\theta_1, \theta_2, \theta_3)$, con $\theta_1 = 1.5$, $\theta_2 = 1.0$ y θ_3 de manera que el coeficiente de correlación, $\rho = \frac{\theta_3}{\sqrt{\theta 1 \theta 2}}$ sea aproximadamente igual a 0.50 con el fin de examinar la bondad de las aproximaciones para datos con una correlación media, en este caso θ_3 =0.62.

Para estimar el parámetro θ se emplea el método de máxima verosimilitud como se describe en Kocherlakota y Kocherlakota (1992, pp. 103-105) [29]. Luego, se aproximan los p-valores bootstrap, p*, del test $R_{n,w}(\hat{\theta}_n)$, para ello, en este caso, ya que $\theta_1 \neq \theta_2$, se consideró la función de peso $(a_1, a_2) = (0, 1)$ para el test, con el fin de examinar el efecto de dar un peso diferente a cada uno de los componentes cuando estos tienen distintas esperanzas y se generan B = 500 muestras bootstrap.

El procedimiento anterior se repite 1000 veces y calcula la fracción de los p-valores estimados que resulten ser menores o iguales que 0.05 y 0.10, que son las estimaciones de las probabilidades del error tipo I para $\alpha = 0.05$ y 0.10.

Si las aproximaciones consideradas fuesen exactas, entonces los p-valores calculados deberían ser una muestra aleatoria de una distribución uniforme en el intervalo (0, 1).

Por lo tanto, para medir la bondad de las aproximaciones consideradas, se calcula el p-valor del test estadístico de uniformidad de Kolmogorov-Smirnov (KS) para cada conjunto de 1000 p-valores obtenidos para el test estadístico.

6.2. Resultados de las Simulaciones del test estadístico $R_{n,w}(\hat{\theta})$, para la probabilidad del error tipo I

En este apartado se muestran los resultados de las simulaciones del test estadístico $R_{n,w}(\hat{\theta})$, para la probabilidad del error tipo I, las cuales se llevarán a cabo como se mencionó en la sección 6.1 para el caso $E(X_1) \neq E(X_2)$, $\rho \approx 0.50$, y los tiempos de procesamiento de cada simulación.

Las Tablas 3-6 resumen los resultados obtenidos, por fines de documentación en las tablas que se muestran a continuación se omiten las salidas de los mil p-valores que entrega cada simulación, requieren de mucho espacio para su presentación y su análisis detallado no es de interés directo para el estudio. En dichas tablas, se denota por $R_{n,a}$ al test estadístico $R_{n,w}(\hat{\theta})$, cuando la función de peso w toma la forma dada por (3), para algún $a = (a_1, a_2)$, en este estudio $(a_1, a_2) = (0, 1)$.

Observando los valores dados en las tablas, se concluye que el bootstrap sin paralelizar y paralelizado con distintos número de workers, proporciona una aproximación precisa a la distribución nula de $R_{n,w}(\hat{\theta})$, en todos los casos tratados con la muestra original mencionada en la sección 6.1.

Respecto al tiempo de ejecución transcurrido para cada simulación, se concluye que al aplicar el método de programación paralela foreach con el backend DoParallelSNOW, la combinación de backend y workers para la simulación que obtuvo la mayor reducción de tiempo en este estudio fue la que se muestra en la tabla 5, lo cual tiene sentido considerando que la simulación.

Al comparar los tiempos de la simulación del test $R_{n,a}$ sin paralelizar, mostrados en la tabla 3, con el tiempo de la simulación que se muestra en la tabla 5, se puede observar una reducción de tiempo de 4544.569 segundos lo cual equivale a 1 hora 14 min y 45 segundos.

Tabla 3: Resultados de simulación del test $R_{n,a}$ sin paralelizar.

Tiempo transcurrido (elapsed)	28757.01
Tamaño muestral	30
Tamaño de muestras Bootstrap	500
Repeticiones Monte Carlo	1000
Estimación de la probabilidad del error tipo I con $\alpha = 0.05$	0.042
Estimación de la probabilidad del error tipo I con $\alpha =$0.1	0.087
Test Kolmogorov Smirnov	0.41315

Fuente: Elaboración propia.

Tabla 4: Resultados de simulación del test $R_{n,a}$ paralelizado con foreach y el backend DoParallelSNOW registrado, corriendo con 16 workers.

Tiempo transcurrido (elapsed)	25974.856
Tamaño muestral	30
Tamaño de muestras Bootstrap	500
Repeticiones Monte Carlo	1000
Estimación de la probabilidad del error tipo I con $\alpha = 0.05$	0.041
Estimación de la probabilidad del error tipo I con $\alpha =$0.1	0.089
Test Kolmogorov Smirnov	0.41315

Fuente: Elaboración propia.

Tabla 5: Resultados de simulación del test $R_{n,a}$ paralelizado con foreach y el backend DoParallelSNOW registrado, corriendo con 4 workers.

Tiempo transcurrido (elapsed)	24212.441
Tamaño muestral	30
Tamaño de muestras Bootstrap	500
Repeticiones Monte Carlo	1000
Estimación de la probabilidad del error tipo I con $\alpha = 0.05$	0.04
Estimación de la probabilidad del error tipo I con $\alpha =$0.1	0.088
Test Kolmogorov Smirnov	0.459543

Fuente: Elaboración propia.

Tabla 6: Resultados de simulación del test $R_{n,a}$ paralelizado con foreach y el backend DoParallelSNOW registrado, corriendo con 3 workers.

Tiempo transcurrido (elapsed)	24365.386
Tamaño muestral	30
Tamaño de muestras Bootstrap	500
Repeticiones Monte Carlo	1000
Estimación de la probabilidad del error tipo I con $\alpha = 0.05$	0.043
Estimación de la probabilidad del error tipo I con $\alpha =$0.1	0.085
Test Kolmogorov Smirnov	0.508494

Fuente: Elaboración propia.

7. Conclusión

En cuanto a los resultados del test estadístico como tal, se pudo probar una vez más que funciona correctamente en todos los casos tratados como menciona en la sección 6.2. y en base al tiempo pese a no ser lo esperado como lo que señala la literatura, se obtuvieron mejoras de tiempo de aproximadamente una hora y media, del programa paralelizado con foreach y el backend DoParallelSnow, versus el programa sin paralelizar.

Del método doMPI se puede decir que si bien logra muy buenos resultados en cuanto a reducción de tiempo de respuesta, cuando funciona del todo bien. Requiere de un nivel de conocimiento computacional y de tiempo para poder poner en práctica los pasos básicos a seguir intentando resolver todos los problemas que se presenten en el camino, pues funcionando en redes de computadoras siempre parece presentar desafíos.

Lamentablemente en este estudio no se pudo llevar a cabo la implementación del paquete multicore para programar junto a foreach pues en el clúster en que se llevaron a cabo las simulaciones correspondientes, no se pudo instalar multicore por problemas de actualización. Sin embargo se realizaron intentos utilizando doMC pero los resultados no fueron los esperados ya que sólo funcionó la ejecución en modo secuencial, lo cual no produjo disminución del tiempo de simulación.

Como conclusión general, se llegó a que las técnicas hoy existentes de paralelización mediante R, son muy útiles en términos de reducción de tiempo de simulación y una vista más clara de los códigos,esto, cuando los programas no presentan dependencias que limiten ejecutar el programa en partes separadas, pues esto produce algunas limitaciones.

El utillizar 4 trabajadores logró reducir el tiempo de simulación del programa paralelizado, ejecutándose en menor tiempo que con más trabajadores, lo cual tiene sentido producto de la sobrecarga que implica el enviar cada tarea, pues el aumento en el número de máquinas o núcleos, no significa automáticamente un gran aumento

de velocidad, porque hay que tomar en cuenta los gastos generales en el envío.

La posibilidad de haber podido practicar la programación paralela aplicada a la simulación del test de bondad de ajuste $R_{n,a}$, mediante plataforma de NLHPC, que son técnicas y plataformas computacionales respectivamente, innovadoras y de última generación ambas, me permitió ganar experiencia en temas, en lo personal, prácticamente nuevos y desconocidos, con lo cual permite que en una ámbito laboral pueda enfrentarme a situaciones similares contando con una base previa.

De los resultados obtenidos en cuanto a reducción de tiempo, pese a no ser tanto como lo esperado, se puede considera un buen punto de partida para una investigación más profunda, a un nivel de conocimiento computacional más elevado.

Como recomendación se propone en el caso de definitivamente querer utilizar la programación paralela mediante el lenguaje de programación R, se realice una reestructuración del programa en estudio, de ser posible hacerlo sin afectar en sus resultados, sin embargo se debe tener presente que puede involucrar mucho tiempo en estudiar e implementar la paralelización y no siempre llegar a buen término. Por lo anterior, de no ser necesario utilizar estrictamente el lenguaje R, se sugiere reescribir el programa en un lenguaje de programación como C, pues al estar más próximo al lenguaje de máquina, o nativo (binario), podría llevar a cabo las simulaciones del programa en menos tiempo que el programa original e incluso que en el paralelizado, lo cual fue sugerido por el Doctor en Ciencias de la Computación, Rubén Carvajal Schiaffino, de la Universidad de Santiago de Chile mientras dictaba una capacitación de programación en paralelo, para el Departamento de Estadística en la Universidad del Bío-Bío el año 2016. Dicha recomendación no se consideró como primera opción, para probar las posibilidades disponibles en R aplicado al programa del test $R_{n,a}$.

8. Referencias Bibliográficas

Referencias

[1] Baringhaus, L. y Henze, N. (1992). A goodness of fit test for the Poisson distribution based on the empirical generating function, *Statistics & Probability Letters*, 13, 269-274.

[2] Calaway, R. y Weston, S. (2015),Package 'doMC'.Foreach Parallel Adaptor for 'parallel'.
Disponible en https://cran.r-project.org/web/packages/doMC/doMC.pdf

[3] Calaway, R. Weston,S. (2015).Package 'foreach'.Provides Foreach Looping Construct for R.
Disponible en https://cran.r-project.org/web/packages/foreach/foreach.pdf

[4] Calaway,R. y Weston,S. Tenenbaum.D Package 'doParallel'. Foreach Parallel Adaptor for the 'parallel'Package.
Disponible en https://cran.r-project.org/web/packages/doParallel/doParallel.pdf

[5] Calaway,R. (2015). Using The iterators Package.
Disponible en https://cran.r-project.org/web/packages/iterators/vignettes/iterators.pdf

[6] Carmona, F. 2007 Curso básico de R-bn
Disponible en http://www.ub.edu/stat/docencia/EADB/Curso basico de R-bn.pdf

[7] Crockett, N. G., (1979). A quick test of fit of a bivariate distribution. In D. McNeil (ed.), *Interactive Statistics*, 185-191. Amsterdam: North-Holland.

[8] Eddelbuettel, D. (2017). Package 'digest'.Create Compact Hash Digests of R Objects.
Disponible en https://cran.r-project.org/web/packages/digest/digest.pdf

[9] Hayes,T. (2014). Parallel Computing in R using the snowfall Package. *Disponible en https://hpcc.usc.edu/files/2014/07ParallelComputingRSnowfall.pdf*

[10] *https://eric.univ-lyon2.fr/∼ricco/tanagra/fichiers/frTanagraParallelProgrammingR.pdf*

[11] *http://gforge.se/2015/02/how-to-go-parallel-in-r-basics-tips#The_parallel_package*

[12] *http://www.chiark.greenend.org.uk/∼sgtatham/putty/faq.html#faq-what*

[13] *http://cran.r-project.org/web/packages/foreach/index.html*

[14] *http://cran.r-project.org/web/packages/iterators/index.html*

[15] *http://cran.r-project.org/web/packages/snow/index.html*

[16] *https://lignux.com/winscp-accede-desde-windows-por-medio-de-ssh-o-sftp/*

[17] *http://r-statistics.co/Parallel-Computing-With-R.html*

[18] *http://web.archive.org/web/20121024234619/http://www.guia-ubuntu.org/index.php?title=Servidor_ssh*

[19] *http://usuarios.nlhpc.cl/index.php/Página_principal*; revisado el "25/marzo/2017"

[20] *http://www.putty.org/*

[21] Johnson, N. L., Kotz, S. y Balakrishnan, N. (1997). Discrete Multivariate Distributions, *New York: John Wiley & Sons.*

[22] Kabacoff.R R in Action. Data analysis and graphics with R *Disponible en http://kek.ksu.ru/eos/DataMining/1379968983.pdf*

[23] Knaus.J (2010). Developing parallel programs using snowfall. *Disponible en https://cran.r-project.org/web/packages/snowfall/vignettes/snowfall.pdf*

[24] Knaus.J, Porzelius.C, Binder.H y Schwarzer.G (2009).Easier Parallel Computing in R with snowfall and sfCluster
Disponible en https://journal.r-project.org/archive/2009-1/RJournal_2009-1_Knaus+et+al.pdf

[25] Karlis, D. y Ntzoufras, I. (2000). On modelling soccer data, *Student*, **3**, 229-244.

[26] Karlis, D. y Ntzoufras, I. (2003a). Analysis of sports data by using bivariate Poisson models, *The Statistician*, **52**, Part 3, 381-393.

[27] Karlis, D. y Ntzoufras, I. (2003b). Bayesian and Non-Bayesian Analysis of Soccer Data using Bivariate Poisson Regression Models. Disponible en *http://www.docstoc.com/docs/39143794/Bayesianand-Non-Ba-yesian-Analysis-of-Soccer-Data-using-Bivariate*.

[28] Karlis, D. y Ntzoufras, I. (2005). Bivariate Poisson and Diagonal Inflated Bivariate Poisson Regression Models in R, *Journal of Statistical Software*, **14**(10), 1-36.

[29] Kocherlakota, S. y Kocherlakota, K. (1992). Bivariate Discrete Distributions, *New York: Marcel Dekker*.

[30] Loukas, S. y Kemp, C. D. (1986). The Index of Dispersion Test for the Bivariate Poisson Distribution, *Biometrics*, **42**, 941-948.

[31] MANTEROLA.C PINEDA.V , GRUPO MINCIREl.(2008) Valor de "p" y la "significación estadística". Aspectos generales y su valor en la práctica clínica* Interpretation of medical statistics.
Disponible en http://www.scielo.cl/pdf/rchcir/v60n1/art18.pdf

[32] Maher, M, J. (1982). Modelling association football scores, *Statistica Neerlandica*, **36**, 109-118.

[33] Novoa-Muñoz, F. y Jiménez-Gamero, M. D. (2014). Testing for the bivariate Poisson distribution, *Metrika*, **77**(6), 771-793.

[34] Package 'parallel'R-core
Disponible en https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf

[35] R Development Core Team (2000) Introduccíon a R Notas sobre R: Un entorno de programacióon para Análisis de Datos y Gráficos .
Disponible en https://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf

[36] R Development Core Team, R. (2009). A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. Disponible en URL http://www.R-project.org.

[37] Revolution analytics(2015). foreach/iterators User's Guide
Disponible en https://packages.revolutionanalytics.com/doc/7.3.0/win/RevoForeachIterators_Users_Guide.pdf

[38] Rosario, R. (2012)Parallelization in R, Revisited.
Disponible en http://www.bytemining.com/files/talks/larug/hpc2012/HPCinRrev2012.pdf

[39] Rue, H. y Salvesen, ø. (2000). Prediction and retrospective analysis of soccer matches in a league, *Statistician*, 49, 399-418.

[40] Santana.J Farfán.E (2014) El arte de programar en R:un lenguaje para la estadística
Disponible en https://cran.r-project.org/doc/contrib/SantanaElartede_programar_en_R.pdf

[41] Schmidberger.M Morgan.M Eddelbuettel.D Yu.H Tierney.L Mansmann.U(2009). State of the Art in Parallel Computing with R
Disponible en https://www.jstatsoft.org/article/view/v031i01/v31i01.pdf

[42] Técnica de paralelizacion automatica

Disponible en http://www.tesisenred.net/bitstream/handle 10803/5983/08Eap08de14.pdf?sequence=8

[43] Teetor.P(2011) R cookbook.

Disponible en https://ase.tufts.edu/bugs/guide/assets/R Cookbook.pdf

[44] Tierney.L, Rossini.A, Na Li,Sevcikova.H (2016). Package 'snow'.Simple Network of Workstations

Disponible en https://cran.r-project.org/web/packages/snow/snow.pdf

[45] Tierney.L (2003). Simple Parallel Statistical Computing in R

Disponible en http://homepage.divms.uiowa.edu/∼luke/talks/uiowa03.pdf

[46] Weston.S (2015). Introducción a doMPI.

Disponible en https://cran.r-project.org/web/packages/doMPI/vignettes/doMPI.pdf

[47] Weston.S (2015)Nesting Foreach Loops

Disponible en https://cran.r-project.org/web/packages/foreach/vignettes/nested.pdf

[48] Weston.S (2015).Package 'doMPI'.Foreach parallel adaptor for the Rmpi package.

Disponible en https://cran.r-project.org/web/packages/doMPI/doMPI.pdf

[49] Weston.S (2015)Getting Started with doMC and foreach.

Disponible en https://cran.r-project.org/web/packages/doMC/ vignettes/gettingstartedMC.pdf

[50] Weston.S y Galaway. R (2015).Getting Started with doParallel and foreach.

Disponible en https://cran.r-project.org/web/packages/doParallel/vignettes/ gettingstartedParallel.pdf

[51] Weston.S (2015) Using The foreach Package.

Disponible en https://cran.r-project.org/web/packages/foreach/vignettes/foreach.pdf

[52] Zhou.M (2014)High performance computing in R using doSNOW package

Disponible en http://biostat.mc.vanderbilt.edu/wiki/pub/Main/MinchunZhou/HPC SNOW.rwn.pdf