REVISTA CUBANA
DE TRANSFORMACIÓN DIGITAL

RNPS 2487
Vol. 1 • Nro. 1 • enero-abril 2020 • págs. 119-131

ARTÍCULO DE REFLEXIÓN

Desarrollo distribuido del algoritmo para la detección de factores anómalos locales en Apache Spark

Distributed Development of the Algorithm for the Detection of Local Outlier Factors in Apache Spark

Raynel Roberto Rodríguez Oliva
19lenyar94@gmail.com • https://orcid.org/0000-0003-4767-1277
EMPRESA CORREOS DE CUBA

Lester Guerra Denis
lguerra@ceis.cujae.edu.cu • https://orcid.org/0000-0002-6183-8128

Humberto Díaz Pando
hdiaz@ceis.cujae.edu.cu • https://orcid.org/0000-0003-1591-8781
UNIVERSIDAD TECNOLÓGICA DE LA HABANA "JOSÉ ANTONIO ECHEVERRÍA", CUJAE, CUBA

Recibido: 2020-12-30 • Aceptado: 2020-03-16

RESUMEN

El desarrollo alcanzado en las tecnologías de la información y las comunicaciones ha resultado en un crecimiento de los datos almacenados y/o intercambiados electrónicamente. El análisis de estos datos permite extraer conocimientos valiosos, posibilitando así la optimización de servicios. El empleo de técnicas de minería de datos puede resultar muy útil en este tipo de análisis. Siendo la detección de anomalías una de las tareas de gran impacto a nivel mundial. Sin embargo, cuando el volumen de datos almacenado no puede ser procesado por las infraestructuras tradicionales se necesitan otras formas más eficientes de procesar la información. El procesamiento paralelo de la información, permite la ejecución de varios procesos concurrentemente, logrando impresionantes poderes de cálculo. El objetivo de este trabajo es desarrollar el algoritmo para la detección de factores anómalos locales para que sea ejecutado en el cual implementa el modelo de programación MapReduce. Son propuestas dos variantes, la primera es determinista y la segunda es más eficiente que la primera, pero de resultados aproximados. A partir de los experimentos realizados y los resultados obtenidos con las pruebas de hipótesis no paramétricas queda demostrado que las variantes propuestas disminuyen los tiempos de ejecución en relación a su variante secuencial.

Palabras clave: *Apache Spark;* factores anómalos locales; *MapReduce;* procesamiento paralelo.

ABSTRACT

The development achieved in information and communication technologies has resulted in a growth in data stored and/or exchanged electronically. The analysis of this data allows extracting valuable knowledge, thus enabling the optimization of services. The use of data mining techniques can be very useful in this type of analysis. Being the detection of anomalies one of the tasks of great impact worldwide. However, when the volume of stored data cannot be processed by traditional infrastructures, other more efficient ways of processing the information are needed. The parallel processing of the information allows the execution of several processes concurrently, achieving impressive calculation powers. The objective of this work is to develop the algorithm for the detection of local anomalous factors so that it is executed in Apache Spark which implements the MapReduce programming model. Two variants are proposed, the first is deterministic and the second is more efficient than the first, but with approximate results. From the experiments carried out and the results obtained with the non-parametric hypothesis tests, it has been shown that the proposed variants decrease the execution times concerning their sequential variant.

Keywords: *Apache Spark; local outlier factor; MapReduce; parallel processing.*

INTRODUCCIÓN

Con el desarrollo vertiginoso de la informática en los últimos años, las organizaciones almacenan grandes volúmenes de datos, de los cuales han necesitado obtener conocimiento que sea más oportuno y útil, principalmente si este influye directo y positivamente en su desarrollo. La Minería de datos es una de las vías existentes para la obtención de dicho conocimiento, y se define como el proceso de extraer conocimiento útil y comprensible, previamente desconocido, desde grandes cantidades de datos (Orallo, *et al.,* 2004).

De manera general, las tareas de Minería de Datos se clasifican en dos grandes categorías: predictivas y descriptivas (Orallo, *et al.,* 2004). Dentro de esta última se encuentra la detección de anomalías, la cual consiste en encontrar objetos que sean diferentes de los demás. Según la literatura consultada, una amplia gama de técnicas han sido aplicada con tal fin, entra las cuales encontramos: redes bayesiana (Calvo-Valverde *et al.,* 2018), redes neuronales con-

volucionales (López, 2019). Por otra parte, se han utilizado técnicas basadas en algoritmos de agrupamiento (Vadoodparast *et al.,* 2015) y en *Vecino más cercano* (Zhang, *et al.,* 2003).

Uno de los algoritmos para la detección de valores anómalos mediante la técnica de V*ecino más cercano* es el Factor Local de Anomalías (LOF, por sus siglas en inglés) (Breunig, *et al.,* 2000). Su objetivo principal es encontrar objetos de datos anómalos midiendo la desviación local de un objeto determinado, con respecto a sus vecinos (Breunig, *et al.,* 2000). El algoritmo LOF (Breunig, *et al.,* 2000) una vez obtenida la *k* distancia y la *k* vecindad las utiliza para determinar la densidad de cada uno de los objetos del dominio. Finalmente, el índice de anomalía de cada objeto se obtiene a través de la comparación de su densidad respecto a la densidad de sus vecinos cercanos, donde aquellos objetos que tengan una densidad sustancialmente menor que sus vecinos, son considerados como valores atípicos o *outlier* (Xiong, *et al.,* 2013).

Dicho algoritmo es un medio valioso para encontrar comportamientos atípicos en operaciones como: detección de intrusos en redes informáticas, detección de fraude financiero, análisis de datos geográficos, procesos de auditorías, entre otras. La meta es tomar decisiones más efectivas, que lleven a mejores resultados.

Actualmente el número de sistemas que producen datos que son recolectados están creciendo exponencialmente, lo que trae consigo que dichos datos se caractericen por su creciente volumen, alta velocidad a la que son generados y variedad en su estructura y formato. Ante tal escenario los sistemas tradicionales se han visto limitados a almacenar y procesar los mismos; condiciones que ha traído consigo que varios expertos la nombren *Big data* (Hassanien, *et al.,* 2015); (Tiwari, 2018). De Mauro (2016) dió una definición formal*: "Big Data* es el activo de información caracterizado por un alto volumen, velocidad y variedad para adquirir tecnología específica y métodos analíticos para su transformación en valor".

Una de las alternativas para dar solución a dichos problemas es a través de la computación paralela y distribuida, la cual consiste en el uso simultáneo de varios recursos de cómputo para resolver un problema computacional (Barney, 2016). Según la organización de los recursos de memoria, las arquitecturas pueden ser clasificadas en memoria compartida, donde los diferentes elementos de procesamiento comparten un mismo espacio de memoria a través del cual se comunican; y memoria distribuida, donde cada elemento de procesamiento dispone de un espacio de memoria, al cual solo puede acceder él y estos se encuentran interconectados, a través de una red por la cual se comunican. Dentro de esta última categoría uno de los modelos de programación que dan respuesta a los problemas mencionados con anterioridad es *MapReduce* (Dean, *et al.,* 2008).

MapReduce es un modelo de programación distribuida (Dean, *et al.,* 2008) que procesa datos estructurados en pares clave-valor y consta de dos fases: *map* y *reduce* (Miner, *et al.,* 2012). De forma general, los datos del dominio son particionados en subconjuntos disjuntos, los cuales son distribuidos por cada uno de los nodos, estos últimos los procesa de forma paralela cumpliéndose así la fase *map*. Los resultados parciales de dicha fase de cada uno de los nodos son combinados en la fase *reduce* para obtener una solución.

Una de las diversas herramientas existentes que extienden dicho modelo, es *Apache Spark*. Este es un marco de trabajo (Reyes-Ortiz, *et al.,* 2015) de procesamiento en memoria, destinada para ser

rápida y de propósito general (Karau ,*et al.,* 2015); incorpora implícitamente paralelismo de datos y está centrado en una estructura de datos llamada *Resilient Distributed Datasets* (RDD, por sus siglas en inglés). Dicha estructura es de sólo lectura de datos distribuidos, que se mantiene tolerante a fallos.

METODOLOGÍA

Algoritmo para la detección de factores anómalos locales.

El LOF es considerado uno de los algoritmos clásicos en la detección de anomalías (Campos, *et al.,* 2016). A pesar de haber transcurrido dos décadas desde su publicación por (Breunig, *et al.,* 2000), disímiles son los análisis alrededor del mismo y las investigaciones que lo emplean en sus soluciones. Entre las más recientes podemos encontrar (Brinker, *et al.,* 2019); da Silva Galaco, *et al.,* 2020); (Devi, *et al.,* 2016); (Ferdosi, *et al.,* 2019); (Mishra, *et al.,* 2019); (Wang, *et al.,* 2020); (Zhang, *et al.,* 2020).

LOF es un algoritmo para la detección de valores anómalos locales, mediante la técnica de vecino más cercano. Se basa en el concepto de la densidad local, donde la localidad está dada por los *k* vecinos más cercanos, cuya distancia se utiliza para estimar la densidad. A través de la comparación de la densidad local de un objeto a las densidades locales de sus vecinos, se pueden identificar regiones de densidad similar, y los objetos que tienen una densidad sustancialmente menor que sus vecinos, son considerados como valores atípicos o anómalos (Xiong, *et al.,* 2013). Su valor de LOF es definido como se muestra en la ecuación 1.

$$LOF_k(p)=\sum_{o\in V_k(p)} lrd_k(p) * \sum_{o\in V_k(p)} da_k(p,o)$$

LOF: valor de anomalía,
k: número de vecinos cercanos,
lrd: densidad local de accesibilidad,
Vk(p): k vecindad del objeto p,
p: un objeto cualquiera,
o: objetos de la vecindad de p,
da: distancia de accesibilidad.

El índice de anomalía de un objeto *p,* está dado por el promedio entre las razones de las densidades de los vecinos de *p* y la densidad del mismo. Mientras más baja sea la densidad local de accesibilidad de *p,* y más alta la densidad local de accesibilidad sus vecinos, entonces mayor será el valor de anomalía de *p* (Breunig, *et al.,* 2000).

El algoritmo cuenta con tres etapas fundamentales las cuales serán descritas a continuación:

- **Primera etapa:** es la encargada de buscar los *k* vecinos más cercanos de cada objeto a partir del cálculo de distancia entre cada uno de ellos.
- **Segunda etapa:** una vez concluida la etapa anterior, es calculada la densidad local para cada objeto a partir de sus *k* vecinos cercanos, los cuales fueron determinados en la etapa anterior.
- **Tercera etapa:** es la responsable del cálculo del índice de anomalía, el cual no es más que un valor numérico que determina cuán anómalo es cada objeto. Esta etapa inicia una vez finalizada la anterior.

Diseño del algoritmo paralelo con selección de vecinos cercanos todos contra todos

Después del análisis realizado y en correspondencia con la secuencia de pasos que realiza el LOF, es aplicado el paralelismo de datos en cada una de las etapas fundamentales del algoritmo.

Aprovechando las ventajas que brinda *Apache Spark* con los RDD los datos serán divididos y asignados para cada uno de los nodos automáticamente, de forma que a cada procesador le corresponda un subgrupo como resultado de la división. Cada procesador realiza la misma operación sobre el subgrupo que le fue asignado. En consecuencia, se propone una fase *map* para obtener los *k* vecinos de cada objeto. Así como la aplicación de otra fase *map* a continuación de la anterior, donde se calcula la densidad local de accesibilidad para cada uno de los objetos en análisis.

Una vez calculada la densidad local de cada objeto, se aplicaría la última fase *map* en la cual se determina el índice de anomalía de los objetos en cuestión. Manteniendo lo descrito en la versión secuencial, serán devueltos los resultados de dicho algoritmo ordenados de forma descendente por su valor de anomalía. De forma general, se puede concluir que serán aplicadas tres fases *map,* una para cada etapa antes mencionada. En la figura 1 se aprecia el diagrama de actividades el cual representa el diseño descrito anteriormente. Las actividades correspondientes a la primera fase *map* se encuentran en color azul, las de la segunda fase *map* en color verde y las de la tercera fase *map* en color amarrillo.

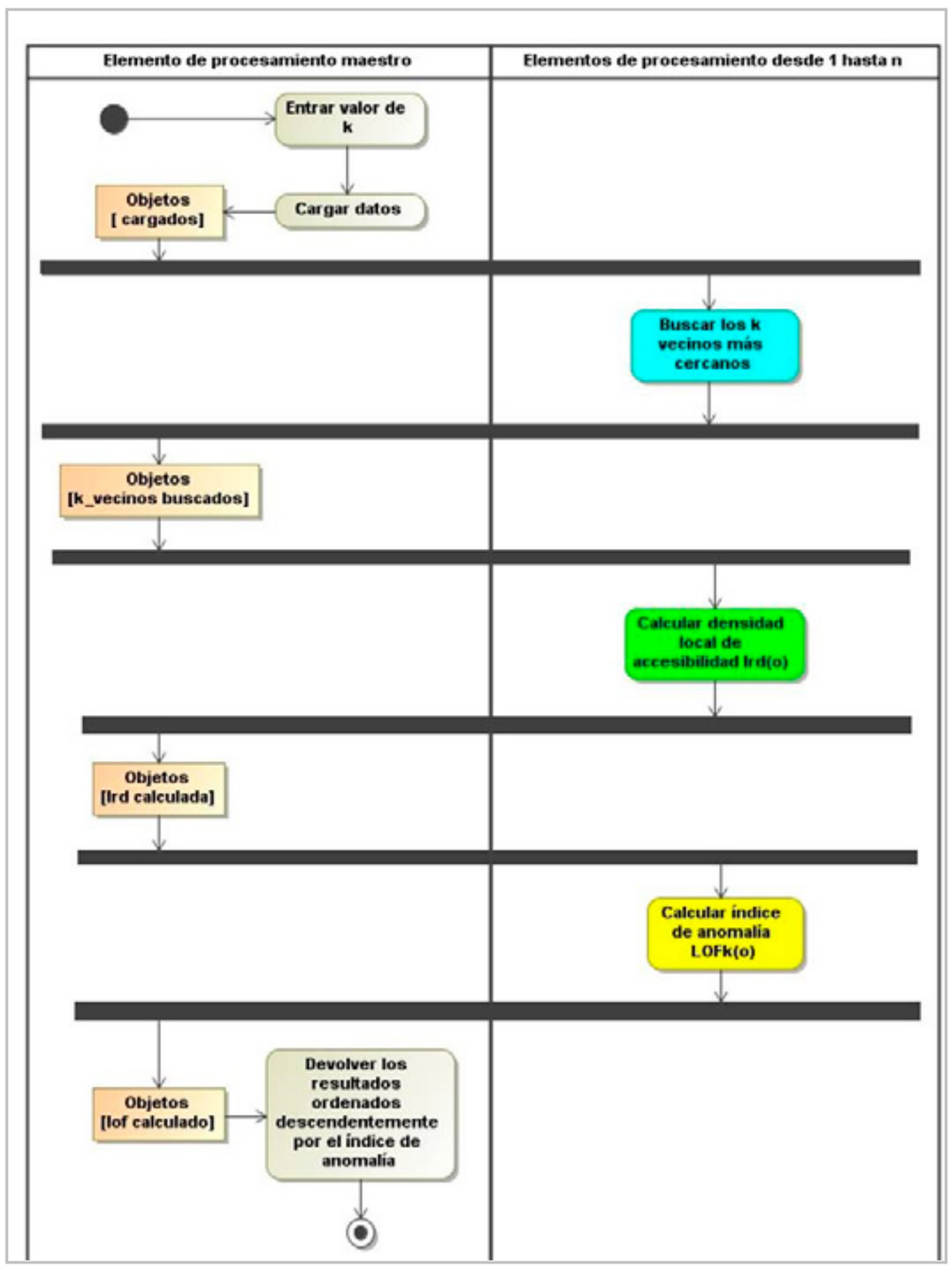


Figura 1. Diagrama de actividades del diseño paralelo todos contra todos.

Diseño del algoritmo paralelo que utiliza pivotes

En los sistemas de memoria distribuida, la comunicación es uno de los factores que influye negativamente en el desarrollo de algoritmos que necesiten información actualizada de otros nodos frecuentemente. El LOF no está exento de dicho tema, dada la necesidad de tener actualizado a cada objeto respecto a sus vecinos cercanos, los cuales pueden no estar en el mismo nodo. Para diseñar una variante en la que se disminuyan los costos de comunicación lo más posible y la eficacia esté lo más próxima a la versión original del algoritmo, se propone la siguiente secuencia de etapas:

1. Entrar el valor de *k* y cargar los datos.
2. Selección de pivotes.
3. Preprocesamiento.
4. Aplicación del algoritmo en cada partición.
5. Devolver los resultados.

Siempre teniendo en cuenta que los resultados serán aproximados, ya que se ha relajado la etapa de búsqueda de vecinos cercanos. Dicho relajamiento consiste en realizar la búsqueda de vecinos cercanos a cada objeto por particiones, por lo que no se tendrán en cuenta aquellos objetos que no pertenezcan a la misma. A partir de esta decisión se verá afectado el índice de anomalía de aquellos objetos que no hayan caído junto a sus vecinos cercanos, y además trae consigo que aumente la eficiencia y disminuya la eficacia. De forma general el algoritmo será aplicado por particiones.

Luego de entrar el valor de *k* y cargar los datos, se debe determinar cuál método de selección de pivotes será ejecutado. Dependiendo del método de selección de pivotes que sea aplicado, serán la cantidad de fases *map* y reduce que se ejecutarán en dicha etapa.

Una vez concluida la selección de pivotes daría paso al Preprocesamiento, el cual comienza con una fase *map* donde se determina por cada partición los objetos más cercanos a cada uno de los pivotes seleccionados en el paso anterior. Seguidamente, con la combinación de una fase *map* y seguido una *reduce*, se calcula la cantidad de objetos por pivotes. Dicho resultado es ordenado ascendentemente por igual criterio. Si el primer pivote contiene más de *k* objetos, se procede a formar tantas particiones como pivotes existan, en caso contrario, los objetos pertenecientes a dicho pivote, en una fase *map* son asignados al siguiente pivote activo más cercano de cada objeto y es recalculada la cantidad de objetos por pivote. Una vez concluida la redistribución es eliminado el pivote que no cumplió con la condición antes mencionada. Finalizada la eliminación antes descrita se procede nuevamente a realizar el mismo procedimiento a los pivotes restantes y así sucesivamente hasta que todos los pivotes cumplan con la condición planteada con anterioridad.

Cuando el Preprocesamiento concluya comenzará la aplicación del algoritmo de forma paralela en cada una de las particiones, en estas últimas son ejecutadas las tres etapas principales del algoritmo. Dichas etapas se ejecutarán una a continuación de la otra, dada la dependencia de datos entre las mismas. Por lo que se propone una fase *map* para cada etapa del algoritmo, dado que se ejecutará por particiones la cantidad de fases *map* ejecutadas será igual a tres por la cantidad de particiones existentes.

Finalizada la ejecución por particiones, se unen los resultados de cada partición. Manteniendo el criterio descrito en la variante secuencial los mismos son ordenados de forma descendente por el valor de anomalía. Una vez que concluya el ordenamiento, los resultados son devueltos.

A partir del tamaño de los diagramas y con el objetivo de no dejar tantos espacios en blanco, a continuación, serán descritos los diagramas de actividades correspondientes a este diseño, los mismos estarán ilustrados una vez que concluya la explicación. De forma general

las actividades que comienzan con: nombre de la actividad y presentan un tridente en la parte inferior derecha de dicha actividad, significa que la misma se define con más detalle en otro diagrama de actividades.

En la figura 2 podemos observar el diagrama de actividades que representa el diseño antes descrito. Seguidamente, en la figura 3 se representa el diagrama de actividades perteneciente a la actividad Preprocesamiento. En este último, en color azul se encuentran las actividades correspondientes a la fase *map* que determina los objetos más cercanos a cada uno de los pivotes, como se mencionó anteriormente el cálculo de la cantidad de objetos consta de dos fases, una *map* identificada en color verde y otra *reduce,* en color violeta. En la figura 4 se puede observar el diagrama de actividades correspondiente a la actividad *Aplicar el algoritmo en cada partición.* En la misma son señaladas en color azul las actividades correspondientes a la primera fase *map,* en color verde las de la segunda fase *map* y en color amarrillo las de la tercera fase *map*. Finalmente, la figura 5 se corresponde con el diagrama de actividades de la actividad *Redistribuir objetos.* En la misma en color azul se encuentran identificadas las actividades correspondientes a la fase *map* encargada de asignar los objetos a su nuevo pivote activo más cercano y la actividad correspondiente a la eliminación del pivote que no cumple con las restricciones planteadas.

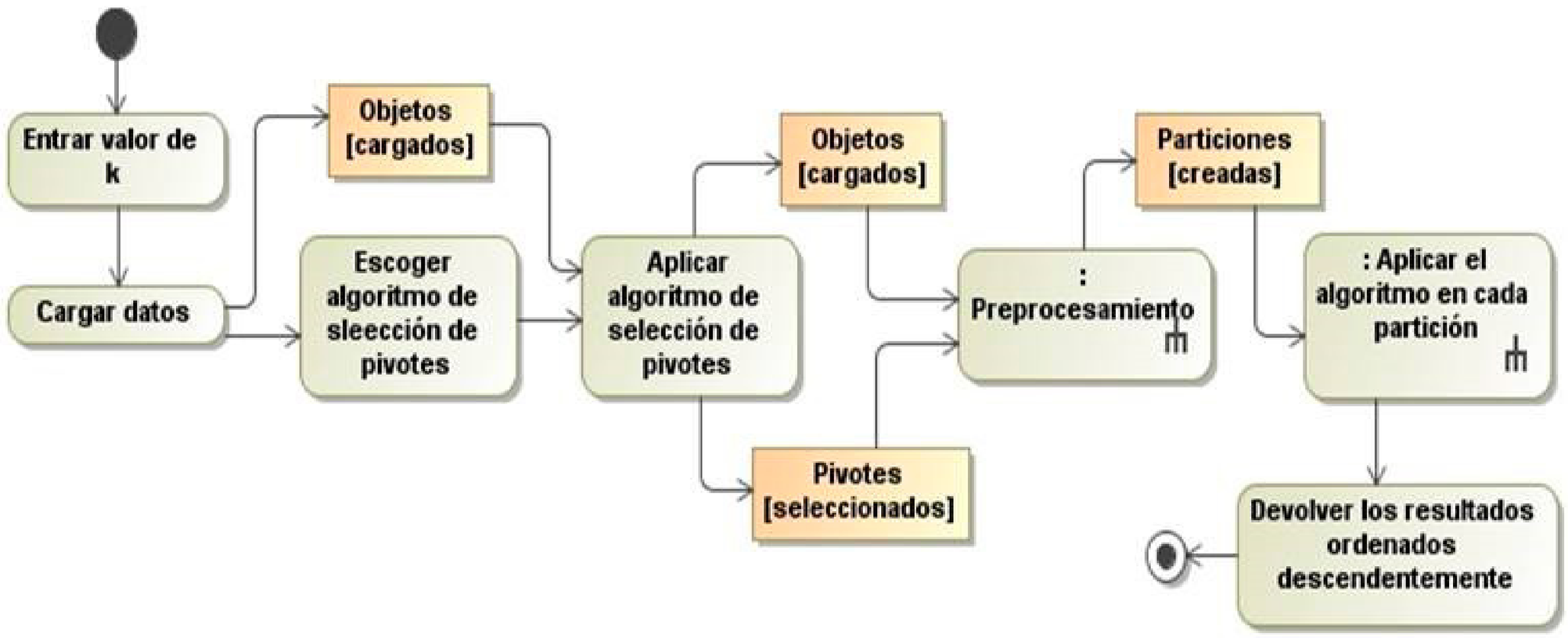
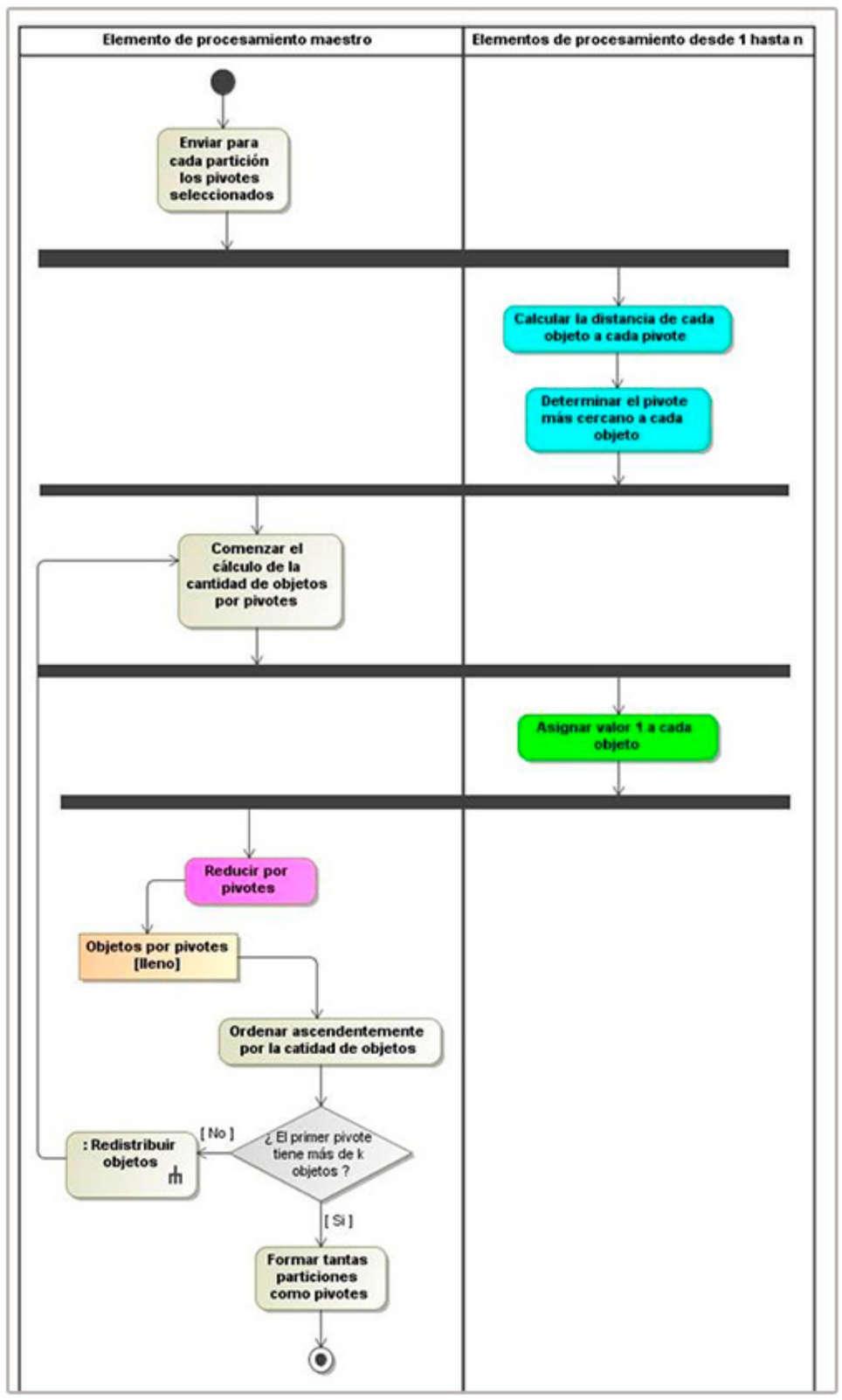


Figura 2. Diagrama de actividades del diseño utilización de pivotes.

RESULTADOS Y DISCUSIÓN

Siguiendo las fases del diseño de experimento, con el objetivo de determinar la configuración de los factores controlables que originan el menor tiempo de ejecución. Se realizaron dos diseños de experimentos ya que en las variantes paralelas presentan un factor más que la variante secuencial. Una vez concluidos dichos experimentos, para realizar comparaciones entre las versiones del algoritmo se realizaron cinco ejecuciones para cada variante utilizando sus configuraciones óptimas. Como configuraciones generales el método de selección de pivotes aplicado fue el Aleatorio, y la cantidad de pivotes a seleccionar es igual a la cantidad



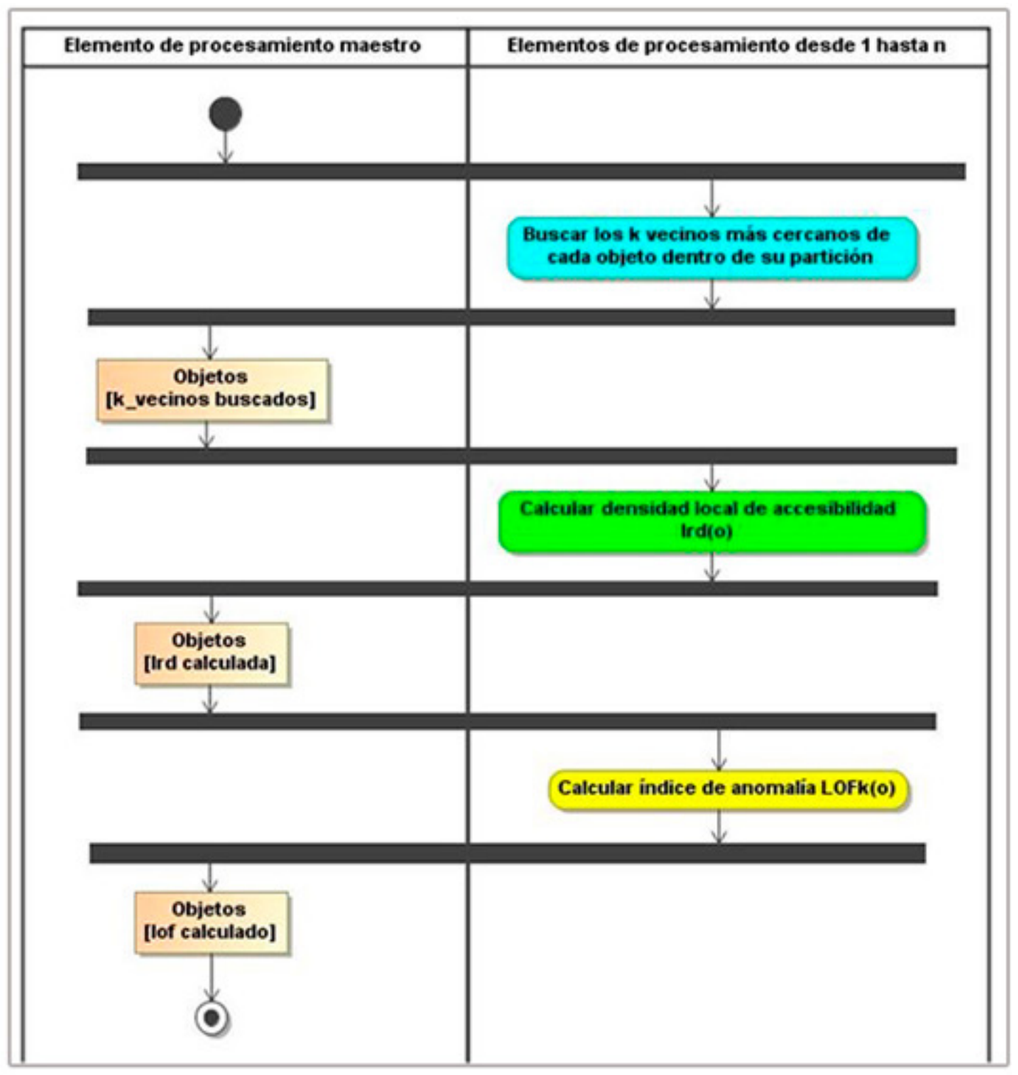


Figura 3 (izquierda). Diagrama de actividades perteneciente a la actividad *Preprocesamiento.*

Figura 4 (arriba). Diagrama de actividades correspondiente a la actividad *Aplicar el algoritmo en cada partición.*

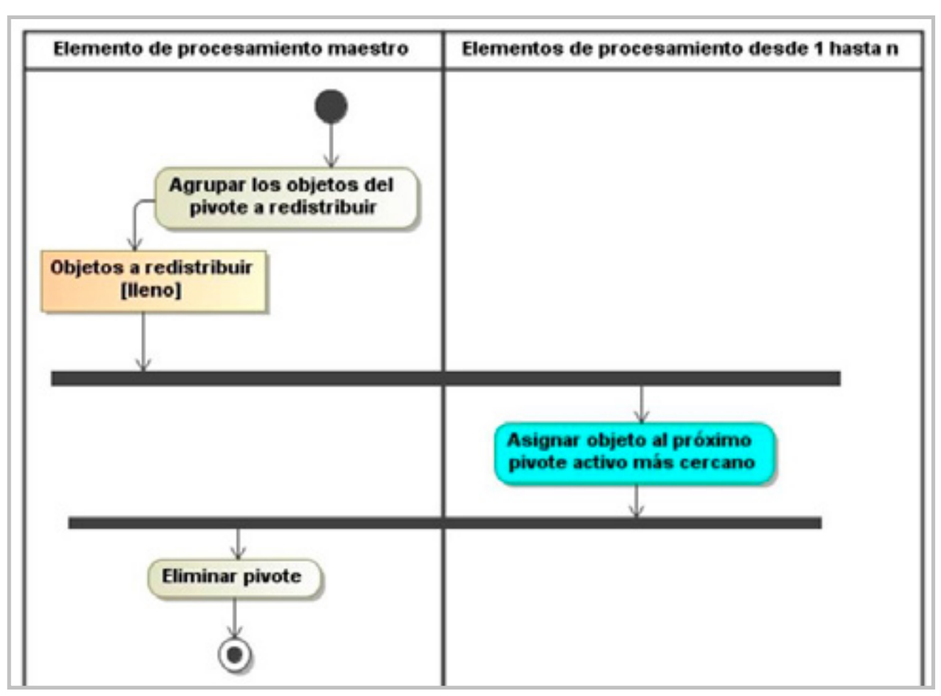


Figura 5 (izquierda). Diagrama de actividades correspondiente a la actividad *Redistribuir objetos.*

de subgrupos en los que fueran divididos los datos. La base de datos experimental cuenta con medio millón de registros y a su vez cada registro con quince atributos a analizar siendo todos numéricos. En la tabla 1 se muestran los tiempos de ejecución (en segundos) para cada una de las ejecuciones.

Con el objetivo de demostrar que las variantes paralelas optimizan los tiempos de ejecución de la variante secuencial se utilizarán pruebas estadísticas.

Tabla 1. Tiempos de ejecución (s) de las variantes del algoritmo utilizando sus configuraciones óptimas.

Ejecución	Secuencial	Todos contra todos	Pivotes
1	2872,434	1904,100	66,742
2	2858,097	1977,779	55,750
3	2898,649	1964,367	75,430
4	2966,742	1974,592	93,483
5	2896,970	1995,369	117,05

Comparación entre la variante secuencial del algoritmo y las variantes paralelas

Una de las pruebas de hipótesis no paramétricas es la de Kruskal-Wallis, la cual es utilizada para determinar si las medianas de dos o más muestras difieren (*Minitab).* La misma será utilizada con el propósito de comprobar si existen diferencias entre los tiempos de ejecución de cada una de las variantes del algoritmo. Para determinar si existen realmente diferencias entre los tiempos de ejecución de las variantes implementadas se definen como hipótesis las siguientes:

- H_0: los algoritmos son iguales en cuanto al tiempo de ejecución.
- H_1: al menos uno de los algoritmos es diferente al resto en cuanto a su tiempo de ejecución.

Donde H_0 es la hipótesis nula y H_1 es la hipótesis alternativa. Se utiliza un $\propto$ (nivel de significación) igual a 0,05. En la figura 6 se muestra el resultado de realizar la prueba usando el *software Minitab 16.* Para efectuar dicha prueba se utilizaron los tiempos de ejecución de la tabla 1.

Al realizar la prueba se obtuvo un valor p= 0,002. Dado que el resultado obtenido es menor que el valor de $\propto$ se rechaza la hipótesis nula y se acepta la hipótesis alternativa, de ahí que se puede concluir que al menos uno de los algoritmos es diferente al resto en cuanto a su tiempo de ejecución. En correspondencia con el resultado obtenido en la prueba de Kruskal-Wallis se pasa a demostrar que la versión secuencial presenta mayores tiempos de ejecución que las versiones paralelas. Una vía para validar dichos resultados es a través de las comparaciones por pares entre la variante secuencial y cada una de las variantes paralelas usando la prueba no paramétrica de Mann-Whitney. Esta última permite determinar si las medianas de dos muestras difieren (*Minitab).*

Prueba de Kruskal-Wallis: Rendimiento vs. Algortimos

Prueba de Kruskal-Wallis en Rendimiento

Algortimos	N	Mediana	Clasificación del promedio	Z
Pivotes	5	75,43	3,0	-3,06
Secuencial	5	2896,97	13,0	3,06
Todos contra todos	5	1974,59	8,0	0,00
General	15		8,0	

H = 12,50 GL = 2 P = 0,002

Figura 6. Resultado de la ejecución de la prueba de Kruskal-Wallis para determinar si existen diferencias entre los tiempos de ejecución de las variantes implementadas.

Una vez que se realizan dos o más comparaciones a un nivel de significancia ∝, la probabilidad de cometer al menos un error de Tipo I (rechazar una hipótesis nula verdadera) es mayor que ∝. Dicha probabilidad puede ser calculada como *1- (1-∝) m* siendo *m* el número de comparaciones realizadas (Camblor, 2012). Por lo que se debe usar algún método de ajuste que garantice que se está respetando el error mínimo de Tipo I fijado con anterioridad (Camblor, 2012). Uno de los métodos es el de Bonferroni. Utilizando dicho método el ∝_*ajustada* que se usará en cada una de las comparaciones será igual a ∝/m. Para cada comparación la hipótesis será rechazada si el valor de *p* obtenido es menor que el ∝_*ajustada*.

En cada comparación realizada entre la variante secuencial y las paralelas se utilizan las siguientes hipótesis:

- H_0: el algoritmo secuencial es igual que la versión paralela en cuanto al tiempo de ejecución.
- H_1: el algoritmo secuencial presenta tiempos de ejecución mayores que la versión paralela.

El ∝ seleccionado para cada una de las comparaciones es igual a 0,05 y el número de comparaciones es dos, luego ∝_*ajustada* = 0,025. Las comparaciones realizadas se muestran a continuación:

- **Variante secuencial contra variante paralela todos contra todos**
  Al realizar dicha comparación en la herramienta *Minitab 16* se obtuvo un valor *p*=0,0061, que es menor que el ∝_*ajustada* por tanto se rechaza la hipótesis nula y se acepta la hipótesis alternativa, por lo que se puede concluir que el algoritmo secuencial presenta tiempos de ejecución mayores que la versión paralela todos contra todos.
- **Variante secuencial contra variante paralela que utiliza pivotes**
  Luego de realizar dicha comparación en la herramienta *Minitab 16* se obtuvo un valor *p*=0,0061, que es menor que el ∝_*ajustada* por lo que se rechaza la hipótesis nula y se acepta la hipótesis alternativa, de ahí que se puede concluir que el algoritmo secuencial presenta tiempos de ejecución mayores que la versión paralela que utiliza pivotes.

Luego de haber terminado las dos comparaciones, se concluye que la variante secuencial presenta mayores tiempos de ejecución que las variantes paralelas.

Comparación entre las variantes paralelas

Con el objetivo de demostrar que la variante paralela todos contra todos presenta mayores tiempos de ejecución que la otra variante paralela, a continuación, se realizará una prueba no paramétrica de Mann-Whitney entre dichas variantes.

- H_0: la versión paralela todos contra todos presenta igual tiempo de ejecución que la versión que utiliza pivotes.
- H_1: la versión paralela todos contra todos presenta mayores tiempos de ejecución que la versión paralela que utiliza pivotes.

El ∝ seleccionado para la comparación es igual a 0,05. La figura 7 muestra el resultado de realizar dicha comparación en la herramienta *Minitab 16*.

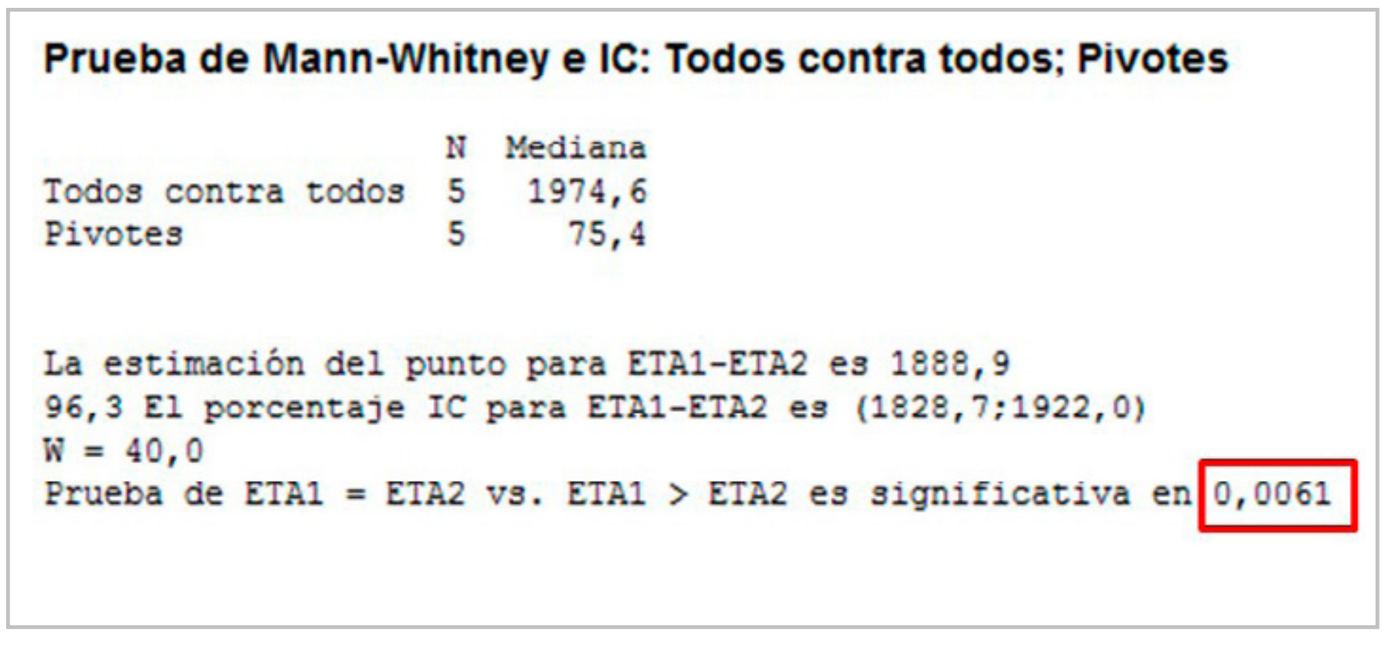


Figura 7. Resultado de la ejecución de la prueba de Mann-Whitney para comparar la variante paralela todos contra todos y la variante paralela distribuida que utiliza pivotes.

Al obtener un valor p=0,0061, que es menor que el valor fijado a ∝ se rechaza la hipótesis nula y se acepta la hipótesis alternativa, de ahí que se puede concluir que la variante paralela todos contra todos presenta mayores tiempos de ejecución que la variante paralela que utiliza pivotes.

Una vez terminadas las pruebas se concluye que la versión paralela todos contra todos presenta mayores tiempos de ejecución que la otra versión paralela.

Métricas de rendimientos

A continuación, se calculan algunas métricas de rendimiento que permiten caracterizar el beneficio obtenido por los algoritmos paralelos. Las mismas se muestran a continuación en la tabla 2.

Tabla 2. Medidas de rendimiento de las variantes paralelas.

	Secuencial	Todos contra todos	Pivotes
Promedio (s)	2898,578	1963,241	81,691
Aceleración	-	1,48	35,48
Eficiencia	-	0,09	2,22

Como muestran los resultados obtenidos la variante que mejor rendimiento presenta es la que utiliza pivotes. La aceleración obtenida para dicha variante es de 35,48 lo que significa es 35,48 veces más rápida que la variante secuencial. Mientras que la eficiencia obtenida de 2,22 implica que se aprovechan en un 222 % más los recursos del procesador respecto a la variante secuencial.

CONCLUSIONES

A partir de los diferentes experimentos realizados y los resultados obtenidos con las pruebas de hipótesis no paramétricas arrojan que las soluciones propuestas disminuyen los tiempos de ejecución en relación a su variante secuencial. Siendo la variante paralela que utiliza pivotes la de mejores tiempos de ejecución.

Se obtuvieron dos diseños paralelos del algoritmo para la detección de factores anómalos locales y como aporte práctico el hecho de que estos diseños disminuyen el tiempo de ejecución del algoritmo en su variante secuencial.

Los resultados de los análisis a los datos correspondientes en las diferentes temáticas mencionadas con anterioridad se obtendrán en un tiempo considerablemente menor.

REFERENCIAS

A. De Mauro, M. J. G., and M. Grimaldi. (2016). A Formal Definition of Big Data based on its essential features. *Library Review,* 65 (3), 122-135.

Barney, B. (2016). Introduction to Parallel Computing. Retrieved 10 de abril de 2017, 2017, from https://computing.llnl.gov/tutorials/parallel_comp/

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). *LOF: identifying density-based local outliers.* Paper presented at the ACM sigmod record.

Brinker, T. J., Hekler, A., Enk, A. H., Klode, J., Hauschild, A., Berking, C., . . . Holland-Letz, T. (2019). Deep learning outperformed 136 of 157 dermatologists in a head-to-head dermoscopic melanoma image classification task. *European Journal of Cancer,* 113, 47-54.

Calvo-Valverde, L.-A., & Acuña-Alpízar, N. J. (2018). Aplicación de métodos agregados en la detección de puntos atípicos en series de tiempo meteorológicas. *Revista Tecnología en Marcha,* 31(1), 98-109.

Camblor, P. M. (2012). Ajuste del valor-p por contrastes múltiples. *Revista chilena de salud pública,* 16(3), p. 225-232.

Campos, G. O., Zimek, A., Sander, J., Campello, R. J., Micenková, B., Schubert, E. & Houle, M. E. (2016). On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery,* 30(4), 891-927.

da Silva Galaco, A. R. B., Freire, R. O., Jesus, L. T., & Serra, O. A. (2020). Experimental and theoretical study of isoreticular lanthanoid organic framework (LOF): Structure and luminescence. *Journal of Luminescence,* 117179.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM,* 51(1), 107-113.

Devi, R. D. H., & Devi, M. I. (2016). Outlier detection algorithm combined with decision tree classifier for early diagnosis of breast cancer. *Int J Adv Engg Tech/Vol. VII/Issue II/April-June, 93,* 98.

Ferdosi, B. J., & Tarek, M. M. (2019). Visual verification and analysis of outliers using optimal outlier detection result by choosing proper algorithm and parameter *Emerging Technologies in Data Mining and Information Security* (pp. 507-517): Springer.

Hassanien, A. E., Azar, A. T., Snasel, V., Kacprzyk, J., & Abawajy, J. (2015). *Big Data in Complex Systems:* Springer.

Karau, H., Konwinski, A., Wendell, P., & Zaharia, M. (2015). *Learning spark: lightning-fast big data analysis:* " O'Reilly Media, Inc.".

López Miguel, P. (2019). Detección de actividades anómalas en espacios públicos mediante redes neuronales profundas.

Miner, D., & Shook, A. (2012). *MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems:* "O'Reilly Media, Inc.".

Minitab. Soporte de Minitab. Retrieved 14 de febrero,2017, 2017, from http://support.minitab.com

Mishra, S., & Chawla, M. (2019). A comparative study of local outlier factor algorithms for outliers detection in data streams *Emerging Technologies in Data Mining and Information Security* (pp. 347-356): Springer.

Orallo, M. J. H., Quintana, M. J. R., Ramírez, C. F., & Schmidt, C. (2004). *Introducción a la Minería de Datos.* Madrid: Pearson Prentice Hall.

Reyes-Ortiz, J. L., Oneto, L., & Anguita, D. (2015). Big data analytics in the cloud: Spark on hadoop vs mpi/openmp on beowulf. *Procedia Computer Science,* 53, 121-130.

Tiwari, H. M. W., Daryanto, Y. (2018). Big Data Analytics in Supply Chain Management between 2010 and 2016: Insights to industries. *Computer and Industrial Engineering,* pp. 319-330.

Vadoodparast, M., & Hamdan, A. R. (2015). Fraudulent Electronic Transaction Detection Using Dynamic KDA Model. I*nternational Journal of Computer Science and Information Security,* 13(3), 90.

Wang, W., Li, H., Wang, K., He, C., & Bai, M. (2020). Pavement crack detection on geodesic shadow removal with local oriented filter on LOF and improved Level set. *Construction and Building Materials,* 237, 117750.

Xiong, Y., Zhu, Y., Philip, S. Y., & Pei, J. (2013). *Towards Cohesive Anomaly Mining.* Paper presented at the AAAI.

Zhang, J., You, H., & Jia, R. (2020). Reliability hazard characterization of wafer-level spatial metrology parameters based on LOF-KNN method. *Microelectronics Reliability, 107,* 113599.

Zhang, Z. M., Salerno, J. J., & Yu, P. S. (2003). *Applying data mining in investigating money laundering crimes.* Paper presented at the Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining.