EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Mejora de un algoritmo de agrupamiento mediante el paradigma de programación paralela

presentada por
Ing. Nancy Salgado Antunez

como requisito para la obtención del grado de
Maestra en Ciencias de la Computación

Director de tesis
Dr. Joaquín Pérez Ortega

Codirector de tesis
Dr. José Ma. Rodríguez Lelis

Cuernavaca, Morelos, México. 22 Mayo de 2023

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

EDUCACIÓN
SEP TecNM CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
RECIBIDO
25 ABR 2023
SUBDIRECCIÓN ACADÉMICA

TECNOLOGICO NACIONAL DE MÉXICO

Centro Nacional de Investigación
y Desarrollo Tecnológico
Departamento de Ciencias Computacionales

Cuernavaca, Mor., 21/abril/2023

OFICIO No. DCC/066/2023
Asunto: Aceptación de documento de tesis
CENIDET-AC-004-M14-OFICIO

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial de Nancy Salgado Antunez, con número de control M20CE047, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis de grado titulado **"MEJORA DE UN ALGORITMO DE AGRUPAMIENTO MEDIANTE EL PARADIGMA DE PROGRAMACIÓN PARALELA"**, y hemos encontrado que se han atendido todas las observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

Joaquín Pérez Ortega
Director de tesis

José María Rodríguez Lelis
Codirector de Tesis

José Crispín Zavala Díaz
Revisor 1

Alicia Martínez Rebollar
Revisor 2

c.c.p. Silvia del Carmen Ortiz Fuentes. Depto. Servicios Escolares
Expediente / Estudiante
MYHP/ibm

EDUCACIÓN
25 ABR 2023
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SERVICIOS ESCOLARES
REC
EBN

Cuernavaca, Mor., 25/abril/2023
No. De Oficio: SAC/055/2023
Asunto: Autorización de impresión de tesis

NANCY SALGADO ANTUNEZ
CANDIDATA AL GRADO DE MAESTRA EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"MEJORA DE UN ALGORITMO DE AGRUPAMIENTO MEDIANTE EL PARADIGMA DE PROGRAMACIÓN PARARELA"**, ha informado a esta Subdirección Académica, qué están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Conocimiento y Tecnología al Servicio de México"

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SUBDIRECCIÓN ACADÉMICA

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO

C. c. p. Departamento de Ciencias Computacionales
Departamento de Servicios Escolares

CMAZ/LMZ

Dedicatoria

A Dios, que me permitió disfrutar de vida y fortaleza para conseguir está anhelada meta.

A Sabino, por todo el apoyo, comprensión y aliento en esta etapa de la vida.

A Sofí, por ser mi motor de vida y quien me alienta a superarme día a día.

A Julieta, por traer nuevos retos e ilusiones.

A mi amorosa y paciente madre, quien es mi ejemplo de vida por su bondad y lucha.

A mi director de tesis, Dr. Joaquín Pérez Ortega, por el tiempo y comprensión que me dedicó, para que esta tesis se realizara.

A mi codirector, Dr. José Ma. Rodríguez Lelis, y los miembros de mi comité tutorial: la Dra. Alicia Martínez Rebollar y el Dr. José Crispín Zavala Díaz, por el seguimiento y aportaciones a esta investigación de tesis.

A mis compañeros y amigos, quienes surgieron en el CENIDET, por compartir y sus metas y sueños, recordándome la importancia de la inspiración.

Agradecimientos

¡Oh abismo de riqueza, de sabiduría y de ciencia el de Dios! ¡Cuán insondables son sus designios e inescrutables sus caminos!
Romanos 11:33

Agradezco a Dios, por la vida, por la salud y las bendiciones que me dio para alcanzar una meta más.

También agradezco a CONACYT, por el apoyo económico que hizo posible la realización de la maestría. Al Tecnológico Nacional de México y al Centro Nacional de Investigación y Desarrollo Tecnológico, por ser las instituciones que hicieron posible este logro.

Quiero agradecer al Dr. Joaquín Pérez Ortega por ser el director de esta tesis, al codirector Dr. José Ma. Rodríguez Lelis y a cada uno de los miembros del comité tutorial**:** Dra. Alicia Martínez Rebollar y Dr. José Crispín Zavala Díaz; por cada una de sus observaciones, sugerencias y consejos a lo largo de esta investigación de maestría.

Además, un agradecimiento especial a la Dra. Leticia Sánchez Lima, por su valioso apoyo. Gracias a cada una de las personas con las que compartí momentos que contribuyeron con mi formación académica y profesional.

Resumen

K-Means es uno de los algoritmos de agrupamiento más utilizados debido a su fácil implementación e interpretación de sus resultados. El problema de agrupamiento de *K-Means* es del tipo *NP-Hard*, lo cual justifica el uso de métodos heurísticos para su solución. En consecuencia, su estudio continúa siendo relevante y vigente. A la fecha se han propuesto implementaciones paralelas del algoritmo *K-Means* estándar, algunas sobre un dominio especifico y muy pocas variantes de propósito general. Sin embargo, estas aún están limitadas para la solución de grandes *datasets*. En contraste se propone un nuevo enfoque de solución el cual consistió en seleccionar una variante secuencial campeona del algoritmo *K-Means*, la cual se rediseñó e implementó de forma paralela y distribuida. Al nuevo algoritmo se le denominó *Hybrid O-K-Means* (*HOK-Means*). Para esta propuesta, se analizaron variantes del algoritmo *K-Means* y se seleccionó una altamente eficiente, denominada *O-K-Means,* la cual mostró tener buen desempeño en tiempo y mínima reducción en la calidad de la solución.

Para validar los resultados de *HOK-Means*, se diseñaron e implementaron diversos experimentos, solucionando tanto *datasets* reales como sintéticos de gran tamaño. Con base en los resultados se mostró que en el mejor de los casos para los *datasets* con un indicador de tamaño *ndk* mayor a 16 millones, se logra un *speedup* (*Sp*) de hasta 22.14x más rápido que *O-K-Means*, logrando una eficiencia paralela (*Ep*) de 0.92. En el peor de los casos, se obtuvo un *Sp* de 1.7x, para *datasets* pequeños. Es importante destacar que se resolvieron grandes *datasets* de hasta 102,464,000 objetos y hasta 70 dimensiones.

Finalmente, con base en los resultados obtenidos, es posible afirmar que esta investigación aporta beneficios a distintos usuarios, principalmente a quienes busquen solucionar grandes conjuntos de datos como los que se presentan en *Big Data* aprovechando todos los recursos disponibles de una red de computadoras.

Abstract

K-Means is one of the most widely used clustering algorithms due to its easy implementation and interpretation of its results. The *K-Means* clustering problem is of the NP-Hard type, which justifies the use of heuristic methods for its solution. Consequently, its study continues to be relevant and current. Parallel implementations of the standard *K-Means* algorithm have been proposed to date, some domain-specific and very few general-purpose variants. However, these are still limited for the solution of large datasets. In contrast, a new solution approach is proposed, which consisted in selecting a champion sequential variant of the K-*Means* algorithm, which was redesigned and implemented in a parallel and distributed way. The new algorithm was named Hybrid O-K-Means (*HOK-Means*). For this proposal, variants of the *K-Means* algorithm were analyzed and a highly efficient one, called *O-K-Means*, was selected, which showed good performance in time and minimal reduction in the quality of the solution.

To validate the *HOK-Means* results, various experiments were designed and implemented, solving both real and large synthetic datasets. Based on the results, it was shown that in the best case for datasets with an *ndk* size indicator greater than 16 million, a speedup (Sp) of up to 22.14x faster than *O-K-Means* is achieved, achieving parallel efficiency. (Ep) of 0.92. In the worst case, a Sp of 1.7x was obtained, for small datasets. It is important to highlight that large datasets of up to 102,464,000 objects and up to 70 dimensions were resolved.

Finally, based on the results obtained, it is possible to affirm that this research brings benefits to different users, mainly to those who seek to solve large data sets such as those presented in Big Data, taking advantage of all the available resources of a computer network.

Contenido Página

Lista de Tablas **Página**

Lista de Figuras **Página**

Capítulo 1

Introducción

A veces son las personas de las que nadie puede imaginar nada las que hacen las cosas que nadie puede imaginar.

Alan Turing

1.1 Motivaciones

Actualmente, debido al desarrollo tecnológico, se generan y almacenan grandes volúmenes de datos, los cuales usualmente permiten generar información útil para la toma de decisiones. Sin embargo, el reto para encontrar patrones de información se ha convertido en un trabajo complicado, debido al gran volumen de los datos y las limitaciones de los recursos computacionales actualmente disponibles. Algunas alternativas para resolver esta problemática, se abordan en disciplinas como: la Ciencia de Datos, la Minería de Datos y la Analítica de Datos, las cuales tiene como objetivo encontrar patrones, los cuales apoyen la toma de decisiones basadas en los datos [1-5]. Para lograr dicho objetivo, estas ciencias se apoyan en diversas técnicas, una de las principales es el agrupamiento de datos o *clustering*.

El agrupamiento de datos u objetos, consiste en formar conjuntos de datos de manera que los objetos miembros de un grupo deben ser similares entre sí y diferentes a los de otros grupos [6]. Sin embargo, con conjuntos de datos del tipo *Big Data*, está tarea se ha vuelto compleja y la capacidad del cómputo secuencial suele ser insuficiente [7]. Dadas estas condiciones, una buena alternativa de solución se encuentra en la explotación de las características del procesamiento paralelo que presentan los dispositivos multinúcleos actuales, los cuales pueden ser principalmente, procesadores y tarjetas de procesamiento gráfico [8-11].

El algoritmo de agrupamiento más utilizado es el algoritmo *K-Means*, debido a su sencilla implementación y fácil interpretación de resultados. Actualmente, se usa ampliamente para el reconocimiento de patrones en diferentes disciplinas como: Biología [12], Mercadotecnia[13, 14], Astronomía[15], Genética [16], clasificación de documentos[17, 18] y segmentación de imágenes [19], entre otras. Sin embargo, con el incremento exponencial en la generación de datos digitales, resolver problemas del tipo *K-Means* con complejidad *NP-Hard* se ha vuelto un reto computacional.

1.2 Descripción del problema

Con la presente investigación, se aborda el problema de reducir los tiempos de procesamiento de un algoritmo de agrupamiento, para la solución de grandes *datasets*, mediante un paradigma de

procesamiento paralelo y distribuido. Para lograrlo, se rediseño e implementó una variante del algoritmo *K-Means*, llamada *O-K-Means* [20], la cual se ejecutará de forma paralela y distribuida.

En la Figura 1.1, se muestra gráficamente el problema relacionado a los tiempos de solución de grandes *datasets* utilizando *K-Means*. En el eje *x*, se expresa el tamaño de los *datasets* a resolver, en el eje *y*, su tiempo de solución. La línea roja representa la función de complejidad temporal del algoritmo *K-Means*. Nótese como, a medida que se incrementa el tamaño de los *datasets*, también se incrementa el tiempo de solución. La línea negra, representa la función de complejidad temporal de las variantes secuenciales de *K-Means*, con optimización en tiempo de solución. Su complejidad temporal idealmente es menor a la de *K-Means*. La línea azul, representa la función de complejidad temporal de las variantes secuenciales de *K-Means* con ejecución paralela y distribuida.

El objetivo que se propone en esta investigación, es reducir el tiempo de procesamiento de una variante, rediseñándola e implementándola para que funcione en un entorno paralelo y distribuido.

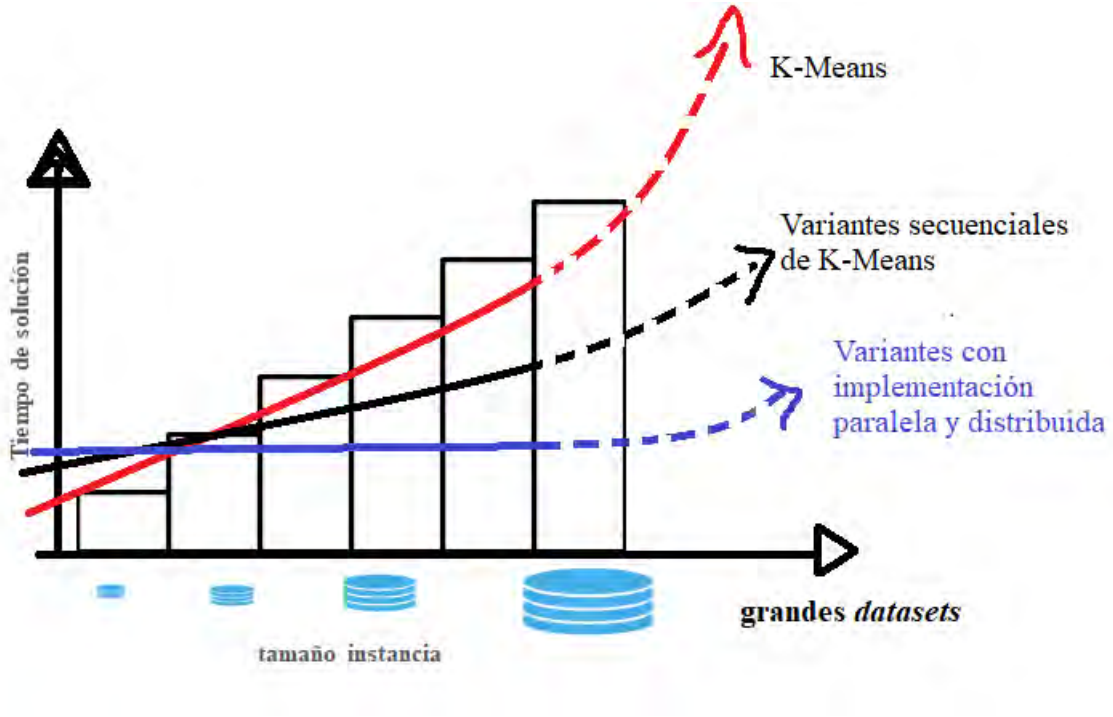


Figura 1.1 Descripción del problema de investigación

1.3 Hipótesis

Es posible reducir el tiempo de solución del algoritmo *O-K-Means* mediante su rediseño e implementación para funcionar de forma paralela y distribuida.

1.4 Objetivo general

Reducir el tiempo de procesamiento del algoritmo *O-K-Means* mediante su rediseño e implementación bajo el paradigma de programación paralela y distribuida.

1.5 Objetivos específicos

1. Implementar el algoritmo de agrupamiento *O-K-Means* con programación paralela y distribuida.
2. Validar los resultados del algoritmo implementado, utilizando *datasets* referenciados en artículos destacados.
3. Evaluar los resultados de tiempo y calidad del agrupamiento empleando el algoritmo implementado.

1.6 Alcances

Al término de la tesis se tendrá una implementación en lenguaje *Python* del algoritmo *O-K-Means*, que se ejecute de forma paralela y distribuida.

1.7 Limitaciones

Las limitaciones de esta investigación fueron las siguientes:

1. La implementación computacional fue probada, en el equipo y *software* disponible en el Centro Nacional de Investigación y Desarrollo Tecnológico.

2. Los experimentos computacionales se realizaron con *datasets* reales del repositorio UCI [21] reconocidos por la comunidad científica. Los *datasets* de prueba sintéticos fueron generados como parte de la investigación.

3. La validación de los resultados se realizó de manera experimental.

1.8 Contexto de la investigación

Desde el año 2005, en CENIDET se han estudiado y optimizando diversos algoritmos de agrupamiento. Entre ellos destacan investigaciones relacionadas con los algoritmos *K-Means y Fuzzy C-Means*. Los avances y resultados obtenidos, han sido reportados en distintas tesis de maestría y de doctorado, así como en diversos artículos científicos publicados en revistas de alto reconocimiento. A continuación, se mencionan algunas de estas publicaciones:

1) Tesis de maestría:

   a) Basave Torres Rosy Ilda, "Mejoramiento de la eficiencia y eficacia del algoritmo de agrupamiento *K-Means* mediante una nueva condición de convergencia", 2005. En esta tesis, se presenta una mejora al algoritmo *K-Means* mediante la implementación de un nuevo criterio de convergencia que garantiza que el algoritmo pare en un primer óptimo local, posibilitando la reducción del número de iteraciones.
   b) Díaz Lorenzo Alber, "Desarrollo de una mejora al algoritmo K-Means orientada al paradigma de Big Data", 2018. En esta tesis se propone *H-Kmeans*, una mejora híbrida que permite procesar, de manera optimizada, un *dataset* dado mediante selección y composición de mejoras secuenciales en diferentes etapas.

2) Tesis de doctorado:

   a) Hidalgo Reyes Miguel Ángel, "Mejora del algoritmo K-Means", 2016. En esta tesis, se presenta una mejora al algoritmo *K-Means* en la fase de clasificación. Con esta mejora, se reduce el número de centroides de grupos que intervienen en los cálculos de distancia objeto-centroide sin afectar de manera significativa la calidad de la solución.
   b) Almanza Ortega Nelva Nely, "Desarrollo de heurísticas para la mejora del algoritmo *K-Means* en las etapas de clasificación y convergencia", 2018. En esta tesis se presentan dos heurísticas mejoradas del algoritmo *K-Means*. Una, *A-Means*, determina de manera temprana la membresía de un objeto a un grupo, excluyéndolo de cálculos posteriores de distancia objeto centroides. La otra heurística, *O-K-Means*, acelera el proceso de convergencia, el algoritmo se detiene, cuando el porcentaje de los objetos que cambian de grupo es menor a un umbral.

3) Artículos:

a) Pérez Ortega Joaquín, Pazos Rodolfo, Cruz Laura, Reyes Gerardo, Basave Rosy, Fraire Hector, "Improving the Efficiency and Efficacy of the K-Means Clustering Algorithm through a new convergence condition", 2007. En este artículo se presenta una nueva condición de convergencia que consiste en detener el algoritmo *K-Means* cuando se encuentra un óptimo local o bien, cuando no hay más cambios de objetos entre grupos.
b) Pérez Ortega Joaquín, Boone Rojas Ma. Del Rocio, Somodevilla García María J., "Research issues on K-means Algorithm an Experimental Trial Using Matlab", 2009. En este artículo se presentan los resultados del análisis de los trabajos más representativos que están relacionados con las investigaciones dedicadas a mejorar el algoritmo *K-Means*. Para analizar los diferentes trabajos, se implementó un *framework* que permite obtener resultados experimentales del algoritmo *K-Means* del paquete de Matlab solucionando *datasets* de datos obtenidas del repositorio UCI.
c) Pérez Ortega Joaquín, Almanza Ortega Nelva Nelly, Romero David, "Balancing effort and benefit of K-means clustering algorithms in Big Data realms", 2018. En este artículo se presentan la heurística *O-K-Means*, desarrollada en la tesis de doctorado de Almanza[20] descrita en el Punto 2. La experimentación y sus resultados son relevantes, para la presente investigación, porque permitieron hacer la comparación entre los resultados de *O-K-Means* secuencial y la propuesta *HOK-Means*.

1.9 Organización del documento

El documento que aquí se presenta, consta de cinco capítulos. En el Capítulo 2, se exponen las principales investigaciones que tienen relación con esta tesis de maestría. En el Capítulo 3, se muestra el enfoque de solución general con el cual se resolvió el problema investigado. En el Capítulo 4, por una parte, se describen los experimentos empleados para la validación de la propuesta de solución y por otra, se muestran los resultados obtenidos. En el Capítulo 5, se exponen las conclusiones derivadas de la presente investigación, así como las propuestas para continuar con nuevas investigaciones sobre mejoras de *K-Means* paralelo y distribuido.

Capítulo 2

Revisión del estado del arte de *K-Means*

Data is a precious thing and will last longer than the systems themselves.

Tim Berners-Lee

En la actualidad pueden encontrarse en la literatura, varios trabajos sobre *K-Means* enfocados en disminuir su tiempo de ejecución con grandes *datasets*. Se han desarrollado tres enfoques principales: a) el desarrollo de algoritmos secuenciales eficientes; b) el paralelismo de algoritmos; y c) la distribución en el almacenamiento y procesamiento de los *datasets*.

En este apartado se presentan brevemente, algunas publicaciones relevantes, que se tomaron como apoyo para la realización de esta tesis. El propósito es, conocer las variantes o mejoras del algoritmo *K-Means*, tanto secuenciales como paralelas y distribuidas. Conocer qué estrategias se aplican para realizar pruebas frente a grandes *datasets* y con ello generar una buena base experimental.

En la sección 2.1 brevemente se abordan trabajos que han comparado mejoras o variantes secuenciales, entre ellas las más prometedoras, entre la cuales se seleccionó la de *O-K-Means*, como base para la variante paralelo y distribuido propuesta. En la sección 2.2 se describen algunas publicaciones de algoritmos paralelos basados en *K-Means*, tanto de propósito general como algunos adecuados a problemas específicos.

2.1 Variantes secuenciales

En el año de 2019, Pérez-Ortega et al. en [22] realizaron una revisión sistemática de las mejoras del algoritmo *K-Means*. En esta se muestra la evolución del algoritmo *K-Means*, desde sus orígenes, se incluye los algoritmos base propuestos por Steinhaus (1956), Lloyd (1957), MacQueen (1967) y Jancey (1966). Además, se describen brevemente 79 mejoras secuenciales, clasificadas por la etapa en la cual se aplica el ajuste. Cabe resaltar que, se mencionan solo ocho algoritmos optimizados en la etapa de convergencia. Con base en el resultado de esta investigación se analizaron las mejoras de *Fahim* y *K-Means++,* y se seleccionó *Fahim* como una de las mejoras a evaluar para esta investigación de tesis.

En el año 2022, Calderón, en [23], evaluó ampliamente de manera experimental dos variantes prometedoras en la solución de grandes *datasets*, *Fahim* y *O-K-Means*. Con las cuales se solucionaron 37 *datasets* reales de repositorios reconocidos, de los cuales 14 fueron catalogadas como grandes. Los resultados obtenidos muestran que la variante *Fahim* es dominante al resolver *datasets* grandes optimizando la calidad de la solución hasta un 0.71%. Mientras la variante *O-K-Means* mostró ser dominante con los *datasets* grandes al resolverlas en menor tiempo hasta un 93.57%. El índice *ndk* más grande con el que se experimentó fue de 290 millones. Cabe destacar

que con las cuatro *datasets* con valor de índice *ndk* más grandes, *O-K-Means* presenta mejores resultados en reducción de tiempo de ejecución y una pérdida de calidad poco significativa y con una desviación estándar menor que *Fahim*.

En el año 2006 en [24], Fahim et al. describió con detalle la variante denominada *Fahim*. Se afirmó que, a partir de la segunda iteración, si la distancia de un objeto x_i al centroide de su grupo $G(j)$ disminuye, con respecto a la iteración anterior, entonces x_i en el ciclo actual, pertenece al grupo $G(j)$ sin necesidad de calcular la distancia desde cada x_i a los centroides restantes. Este ajuste se realizó en la etapa de clasificación.

En el año 2018, en el artículo [6], Pérez-Ortega et al. describió con detalle la variante llamada *O-K-Means*. Novedosa por optimizar el algoritmo en la etapa de convergencia. Esta mejora acelera el proceso de convergencia, deteniendo el algoritmo cuando el porcentaje de objetos que cambian de grupo, en una iteración es menor que un umbral. Se muestra cómo el porcentaje de objetos que cambia en cada iteración es mayor en las primeras iteraciones y conforme se va iterando, el porcentaje de cambio es cada vez más pequeño hasta llegar a cero. El valor del umbral expresa una relación entre el esfuerzo computacional y la calidad de la solución. Por lo tanto, aunque el aumento en la calidad de agrupamiento de la solución es mínima, el algoritmo invierte el mismo esfuerzo computacional para realizar cada iteración. El umbral de parada propuesto disminuye el número de iteraciones conservando la mayor parte de la calidad de la solución del algoritmo *K-Means*. En sus resultados experimentales presentan una reducción promedio del tiempo de ejecución del 93.88 %, con solo una reducción del 0.4 % en la calidad del agrupamiento en comparación con *K-Means* estándar.

2.2 Variantes paralelas

Con el avance tecnológico en los equipos de cómputo, a los cuales se agregan más procesadores, tarjetas gráficas y otros dispositivos aceleradores; una forma de disminuir aún más los tiempos de procesamiento de las variantes, es aplicándoles el paradigma de paralelismo y distribución. La paralelización y distribución permiten utilizar al máximo los recursos de los equipos, sean básicos o sofisticados, en la solución del agrupamiento de grandes *datasets*. Aunque el hardware es un elemento importante en el paralelismo, el software es una parte activa debido a que los programas paralelos son más difíciles de escribir que los programas secuenciales, ya que se requiere que haya una buena comunicación y sincronización entre las tareas que se han paralelizado.

Existe una gran cantidad de investigaciones sobre algoritmos paralelos basados en el algoritmo de agrupamiento *K-Means*, los cuales utilizan los diferentes tipos de tecnologías disponibles en el mercado[25]. Desde que surgieron los primeros procesadores multinúcleo, se han desarrollado algoritmos tipo *K-Means* paralelos, en el año de1989, Li y Fang en [26], describieron algoritmos de agrupamiento paralelizados. Dhillon y Modha en 2002 en [27] analizaron las adversidades, como los costos de comunicación en paralelo. Con la aparición de la Arquitectura Unificada de Dispositivos de Cómputo (CUDA), fue posible desarrollar aplicaciones de procesamiento paralelo haciendo uso de las Unidades de Procesamiento Gráfico (GPU), en el año 2015 Jiménez et al. en [15] analizaron el desempeño correspondiente a la implementación de un algoritmo de segmentación sobre procesadores multinúcleo y GPU's, compararon su desempeño respecto de su algoritmo equivalente con OpenMP y el secuencial.

El algoritmo *K-Means* estándar se ha paralelizado utilizando diferentes plataformas, *frameworks*, arquitecturas y dispositivos. Por ejemplo, en *Map Reduce*[17, 18], *Spark* [28], *Peer to Peer* [29], *GPU's* [30, 31], *Multi-core* [32] y *FPGA* [33, 34], entre otros.

Actualmente algunas investigaciones agrupan y comparan el desempeño de las diferentes herramientas usadas para paralelizar y distribuir los algoritmos de agrupamiento basadas en *K-Means* estándar, como lo hace Fahad en el 2014 en [35].

2.2.1 Orientados a un dominio o problema especifico

Asimismo, existen gran cantidad de investigaciones que involucran mejoras paralelas basadas en *K-Means*, que fueron adaptadas a un dominio o problema específico. A continuación, se describen algunas de ellas.

A Novel Bearing Fault Diagnosis Method Using Spark-based Based Parallel ACO-K-Means Clustering Algorithm [36], 2021. En esta investigación se propuso un algoritmo para optimizar el modelo de diagnóstico de fallas de rodamiento en máquinas rotatorias. Utiliza el algoritmo de agrupamiento de optimización de colonias de hormigas paralelas llamado *(ACO)-K-Means* basado en *Spark*. Sus resultados mostraron una buena precisión en el diagnóstico de fallas, una alta eficiencia en el entrenamiento del modelo en un entorno de *Big Data*, solucionaron *datasets* de hasta 119.80 GB. Usaron la arquitectura sofisticada maestro-esclavo, con un nodo maestro y ocho nodos esclavos. Cada nodo tiene ocho procesadores y 64 GB de memoria. Muestra

un *Sp* de hasta 57.71x más rápido que el *(ACO)-K-Means* serial, con una eficiencia paralela de 0.90.

An Analysis of Distributed Document Clustering Using MapReduce Based K-Means Algorithm [18], 2020. En este artículo se modificó un algoritmo basado en *K-Means* adaptado para agrupamiento de datos de documentos, el cual está basado en *MapReduce* distribuido. Su principal contribución fue mejorar el desempeño del algoritmo en particular se reporta que el mejor resultado fue 4.8x más rápido que la versión secuencial, utilizando 10 nodos, al resolver un *dataset* con un tamaño de 750MB.

An Efficient MapReduce-Based Parallel Clustering Algorithm for Distributed Traffic Subarea Division [37], 2015. En este artículo se presentó un algoritmo *K-Means* aplicado a resolver problemas para división de subáreas de tráfico. El algoritmo llamado *Par3PKM* está basado en *Map Reduce* sobre la plataforma distribuida *Hadoop*. Se presentaron los resultados de la experimentación de tres algoritmos que procesan en paralelo. Se solucionaron *datasets* del repositorio UCI. Se procesaron *datasets* de hasta 2GB. Aplican la arquitectura maestro esclavo, con ocho nodos esclavos y un maestro. Mostró una aceleración de 5x más rápido usando ocho nodos que un solo nodo.

2.2.2 Basadas en una mejora secuencial de *K-Means* de propósito general

Se han realizado muchas mejoras del algoritmo *K-Means* estándar, incluyendo las paralelas y distribuida. Sin embargo, hay pocas versiones paralelas de algoritmos previamente mejorados y de propósito general del tipo *K-Means*. A continuación, se describen tres variantes paralelizadas.

1. *High performance parallel k-means clustering for disk-resident datasets on multi-core CPUs* [32], 2014. Se propuso una variante paralela de *K-Means++,* usando multinúcleo en una sola computadora. En este artículo se presentó la división del conjunto de datos de tal manera que cada procesador trabaja un subconjunto de objetos en paralelo. El máximo *Sp* conseguido fue 7.7x, con un ordenador de 12 núcleos y una eficiencia paralela de 0.64.

2. *Parallel k-means++ for multiple shared-memory architectures*, [38], 2016. Implementaron el algoritmo *K-Means++* en tres arquitecturas diferentes para memoria compartida: CPU multinúcleo, GPU de alto rendimiento y la plataforma Cray XMT masivamente

multiproceso. El objetivo de esta investigación fue mostrar una relación del desempeño de cada plataforma con el número de objetos, atributos y grupos. Con base en los rangos de valores para *n, d* y *k* hacen una predicción de cuál de sus plataformas sería la más rápida para cada caso. Mientras GPU funcionó mejor con menos *n* y más *d*, CPU multinúcleo funciono mejor con mayor *n* y mayor *d*.

3. *The parallelization and optimization of K-means algorithm based on spark* [28], 2020. Se propuso una mejora en la fase de inicialización del algoritmo *K-Means* denominado *Canopy*. Esta mejora consistió en seleccionar el conjunto de centroides utilizando el método de densidad ponderada para reducir el impacto de los valores atípicos en los resultados de la agrupación. El mejor resultado es una *Sp* de 3x y una eficiencia paralela de 0.5. El tamaño del conjunto de datos más grande es de 1.72 GB. El algoritmo se paralelizó utilizando la plataforma *Spark* con ocho núcleos.

En la Tabla 2.1, se presenta un resumen de las principales características mencionadas explícitamente en artículos, sobre implementaciones paralelas y distribuidas del algoritmo *K-Means* y sus variantes. En la primera columna, se muestra el nombre utilizado para el algoritmo, el año de publicación y la cita del artículo. En la segunda, si el algoritmo esta basado en una variante o no (S=si y N=no). En la tercera, si es distribuido. En la cuarta, el propósito del algoritmo, G =general o E= especifico. En la quinta, la plataforma que utilizaron para paralelizar. En la sexta, la arquitectura computacional que utilizaron (ME=Maestro Esclavo; JF= Join and Fork) y la etapa del algoritmo en la que se hizo la optimización (Cl=Clasificación, I= Inicialización). En la séptima, las características del *hardware* utilizado. En la octava las características del *dataset* más grande que se utilizó, algunas publicaciones tienen como unidad *n*, que es el número de objetos, la *d* que representa el número de dimensiones y *k*, que representa el número de grupos y otras están en tamaño del archivo. En la novena, cantidades de *datasets* utilizados (R=reales, S=sintéticos), por ejemplo “2R, 6S” son 2 *datasets* reales y 6 sintéticos. En la décima el mejor *Speedup* obtenido. En la onceava, la mejor Eficiencia paralela obtenida. La simbología “---” indica no especificado en el artículo.

Tabla 2.1 Resumen de artículos sobre implementaciones paralelas y distribuidas del algoritmo *K-Means* y sus variantes

Algoritmo	Características de algoritmo			Paralelismo		Características de HW	*Dataset* mayor		Métricas	
	V	D	P	Plataforma: Fase PD	Arq		Rasgos	Tipo	*Sp*	*Ep*
ACO-K-Means, 2021 [36]	S	S	E	Spark, Hadoop: Cl	ME	8 equipos: Intel® Core™ i7 9700K, 64 GB RAM. Drive, port speed 1000Mbps 1 equipo: Intel Xenon E3-1225 v5, 64 GB RAM. Drive, port speed 1000Mbps	119Gb	3R	51.71	0.79
K-Means, 2020 [18]	N	S	E	MapReduce, Hadoop: Cl	ME	10 equipos: Intel Core 2 Duo, 8 GB RAM; 80 GB HD. Ethernet IPV4 LAN speed of 100 Mb/s.	1GB	4R	4.6	0.11
K-Means, 2020[39]	N	N	G	---	---	Supercomputadora *Sunway TaihuLight*	*n*=1.3e6 *k*=160,000 *d*= 196,608	5R	--	--
MR-MDBO, 2020 [40]	S	S	G	MapReduce, Hadoop: Cl	ME	10 equipos: Intel Core-i7 con 3.20 GHz, 16 GB RAM, 2 TB HD	*n*=7,062,606 *k*=10 *d*= 115	5R	--	--
K-Means, 2020 [28]	S	S	G	Spark: Cl	ME	4 equipos: CPU dual core 2.5 GHz, 4 GB RAM, 500 GB HD	1.72 GB *n*=13,226,2717	8R	3	0.5
K-Means, 2019[17]	S	S	E	MapReduce, Hadoop: Cl	ME	4 equipos: Intel i3 4 cores of 3.07 GHz, 1.5 GB RAM; 500 GB HD	20mil documentos	--	2.8	0.35
K-Means, 2019 [12]	S	S	E	GPU y MPI	ME	GPU: Intel Core i5 2.7 GHz, 16GB RAM, Nvidia GeForceGT 640M with 512 MB. MPI: supercomputer cluster SGI Altix (CESUP).4, 64 GB RAM, 36 core AMD	--	7R	GPU: 203.45 MPI: 33	0.91
K-Means, 2019 [19]	S	S	E	MPI	ME	High Performance Computing (HPC) cluster	imagen: 3000 * 1670	1R	69.84	0.72

Algoritmo	Características de algoritmo			Paralelismo		Características de HW	*Dataset* mayor		Métricas	
	V	D	P	Plataforma: Fase PD	Arq		Rasgos	Tipo	*Sp*	*Ep*
							pixeles; *k*=64			
K-Means, 2018 [31]	S	N	G	GPU	---	Intel Core i7-6700K (“Skylake”) CPU with 32 GB memory and an Nvidia GeForce GTX 1080 (“Pascal”) GPU with 8 GB memory on a 2 GB	2GB *d*=4	1S	---	---
HdK-Means, 2017[41]	N	S	G	MapReduce, Hadoop: Cl	ME	Intel Core i7-6700 CPU @ 3.40 GHz x 8, 16 GB memory, 300 GB hard drive, Ubuntu 16.04 OS Intel Core i5-3450 CPU @3.40 GHz x 2, 4GB memory, 300 GB hard drive and Ubuntu 16.04 OS. An Ethernet switch	*n*=20,000 *d*=8,000 *k*=2500 357MB	2S	1.7	--
MPKM PGKM, K-Means, 2016[16]	S	S	E	Multi-core: Cl	JF	Platform 1 Intel® Pentium® D Processor 935, Windows7, 2 cores Platform 2 Intel® Core™i5-4440 Processor, Windows7, 4 cores Platform 3 Intel® Core™ i7-5960X Processor, Extreme Edition Windows7, 8 cores Platform 4 Intel® Xeon® Processor E5-2698, Windows Server, 16 cores	500K *d*=2	1S	---	---
K-Means++, 2016[38]	S	S	G	GPU y openMP: I y Cl	ME, ---	GPU: Nvidia Tesla C1060 GPU OpenMP: 2 equipos con 2 Intel Xeon quad-core CPUs running at 2.66 GHz Cray XMT supercomputer	*n*=1,000,000 *d*=10 *k*=512	1S	GPU: 103 --- ---	--- --- ---

Algoritmo	Características de algoritmo			Paralelismo		Características de HW	*Dataset* mayor		Métricas	
	V	D	P	Plataforma: Fase PD	Arq		Rasgos	Tipo	*Sp*	*Ep*
K-Means, 2016[30]	N	S	G	GPU	---	Two socket Xeon Haswell 2.60GHz server 16 cores, 128 GB RAM, equipped with 2X NVIDIA K80 GPU cards (4 GPUs).	80K *n*= 855,367	5S	13.5	---
K-Means, 2016[42]	S	S	G	MapReduce y Spark	ME	10 equipos: 3.2GHz Intel Core i5-4460 CPU, 4GB Memory. were connected by 1000M Ethernet	*n*=100100 *d*=10	9S	---	---
SPAB-DKMC, 2015[43]	S	S	G	MapReduce Hadoop: I y Cl	ME	6 equipos	*n*=3,000,000 *d*=13 *k*=3	2R 6S	---	--
Par3PKM, 2015 [37]	S	S	E	MapReduce Hadoop: I y Cl	ME	8 equipos: Intel Xeon E7-4820 2.00GHz CPU (4-core) and 8.00GB RAM.	2560MB	6R 6S	6.4	0.8
K-Means, 2015[15]	S	N	E	GPU y OpenMP	JF	CPU core i7 de 8 core. NVidia GeForce GT 610, esta posee 48 cores, memoria global de 1024 MB, memoria compartida de 48 KB por bloque.	322MB *n*=3,000,000	4R	GPU= 16 Open MP = 1.7	Open MP=0 .21
k-means++ y KD-tree, 2014 [32]	S	S	G	Multicore	ME	2 equipos: 6-core Intel Xeon 2.66 GHz processors and 8GB memoria	*n*= 11,620,300 *d*=57 1,583.8MB	6R 1S	7.7	0.64
PK-Means, 2009[44]	N	S	G	MapReduce	ME	4 equipos: con 2 core a 2.8 GHz y 4GB RAM.	8GB	5S	3.3	0.41

Algoritmo	Características de algoritmo			Paralelismo		Características de HW	*Dataset* mayor		Métricas	
	V	**D**	**P**	**Plataforma: Fase PD**	**Arq**		**Rasgos**	**Tipo**	***Sp***	***Ep***
K-Means, 2002 [27]	N	S	G	MPI	---	IBM SP2 con 16 nodos. Cada nodo es una IBM POWER2: 160 MHz with 256 RAM. Un High-Performance Switch con adaptadores HPS-2.	$n=2^{21}$ k=16 d=8	5S	15.65	0.97
k/h-means, 1999 [9]	S	S	G	---	---	32 equipos conectados por 10 MBits Ethernet.	n= 100,000 d=20 k=20	1R	--	---
CLUSTER, 1998[8]	S	S	G	----	---	10 workstations Sun Sparc. Y una supercomputadora IBM SP-2,	n=262,144 d=45 k=16	7R	---	---

Aunque ya existen algoritmos que paralelizan las variantes de *K-Means* de propósito general, la eficiencia que reportan es limitada, y es previsible que la solución de grandes conjuntos de datos requiera mucho tiempo. En contraste, nuestra propuesta es independiente del dominio de aplicación, además está diseñada para grandes conjuntos de datos, como los que se presentan en el *Big Data*. En particular, nuestra propuesta está inspirada en una variante secuencial altamente eficiente, como se asevera en [6]. En este sentido, la propuesta presentada en esta investigación muestra un aumento significativo en el *speedup* y en la eficiencia paralela, superando las investigaciones mencionadas, con características similares, véase Tabla 2.1. Además, esta propuesta es novedosa, debido a que paraleliza las etapas de convergencia y clasificación. Es destacable que la etapa de convergencia es la menos atendida en la literatura.

Capítulo 3

Enfoque de solución

Ciencia sin conciencia no es más que ruina del alma.

Francois Rabelais

De acuerdo con la literatura especializada, se ha demostrado que el problema que resuelve el algoritmo *K-Means* es *NP-hard* para $k \geq 2$ [45, 46]. Es decir, su complejidad computacional es $O(ndki)$, donde n es el número de objetos, k es el número de grupos, d es el número de dimensiones e i el número de iteraciones [1]. Por esta razón, se usan heurísticas mejoradas, así como paradigmas paralelos y distribuidos, como alternativa de solución. En la presente investigación, se propone el rediseño e implementación de la variante *O-K-Means* para ejecutarla de forma paralela y distribuida, con el objetivo de incrementar su eficiencia en el tiempo de solución de grandes *datasets*.

En el presente capítulo se describe el procedimiento con el cual se desarrolló dicho rediseño y su implementación. En la Sección 3.1 se presenta el enfoque general de solución. En la Sección 3.2 describe a detalle la primera fase de la metodología, que es la selección de la variante. En la Sección 3.3 describe la fase 2 de la metodología, la cual describe el rediseño e implementación del nuevo algoritmo. Las fases 3, 4 y 5 se describen en el Capítulo 4.

3.1 Enfoque general de solución

En esta Sección, se presenta la metodología que se siguió para desarrollar la implementación paralela y distribuida de una variante del algoritmo *K-Means*, con la cual se pretende dar solución al problema planteado. En la Figura 3.1, se muestra el proceso con las fases que componen el enfoque de solución. El proceso consta de cinco fases:

1. En la primera fase, se estudiaron, analizaron e implementaron variantes secuenciales del algoritmo *K-Means* y con base en su análisis se seleccionó una variante la cual sirvió como base, véase Sección 3.2. *O-K-Means* fue la variante seleccionada.
2. En la segunda fase, la variante seleccionada se rediseñó e implementó adecuadamente para ejecutarla de forma paralela y distribuida eficientemente. Al algoritmo resultante se le denominó *Hybrid O-K-Means* (*HOK-Means)*.
3. En la tercera, se midieron y compararon los tiempos de solución de la propuesta, contra los tiempos de los algoritmos base y *K-Means* estándar. Con este fin se diseñaron tres experimentos los cuales incluyen la solución de grandes *datasets*, tanto sintéticos como reales.

4. En la cuarta, se analizaron los resultados de la experimentación. Se emplearon métricas del desempeño paralelo, y los porcentajes de reducción de tiempo y calidad de la solución.
5. En la quinta se analizaron las ventajas del nuevo algoritmo. *HOK-Means* presentó buenos resultados con instancias grandes tanto sintéticas como reales.

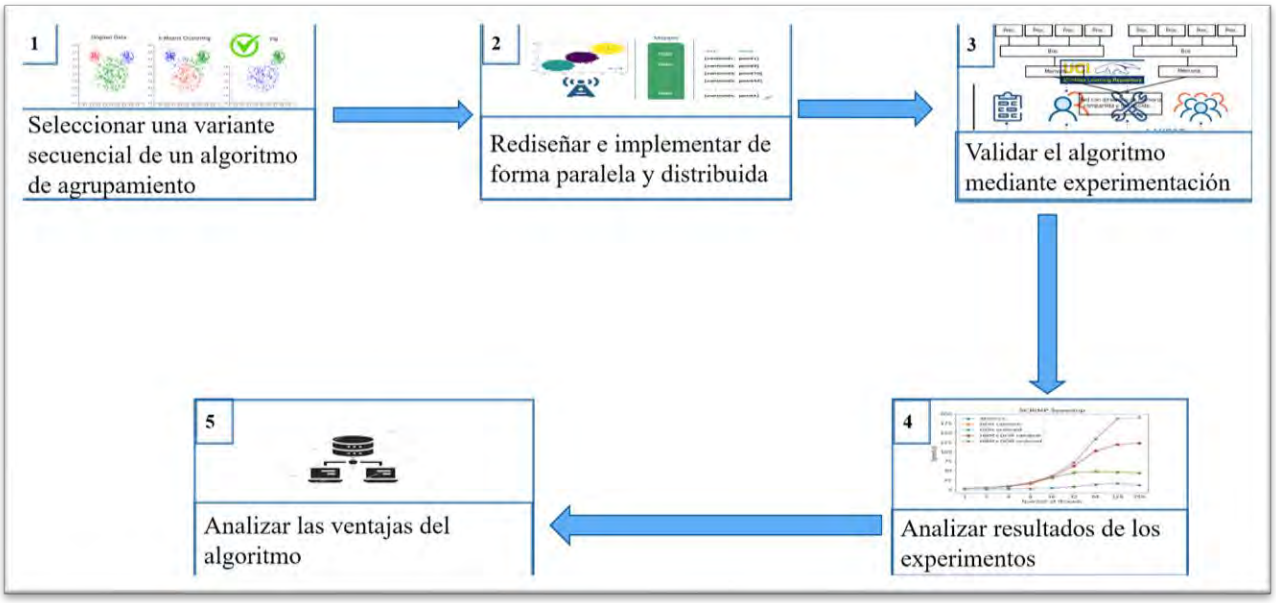


Figura 3.1. Enfoque general de solución

3.2 Selección de *O-K-Means*

El primer paso de la metodología, consistió en elegir una mejora secuencial del algoritmo *K-Means*. De la literatura se estudiaron, implementaron y compararon los algoritmos *O-K-Means*[6, 20], *Fahim* [24] y el hibrido *O-K-Means+Fahim* (*OKM-Fahim*). Para cada algoritmo se resolvieron cuatro *datasets* reales, con dos valores de *k*, cada uno se ejecutó 10 veces, con centroides iniciales diferentes. El valor de la muestra es de 10 debido a que solo se quería corroborar que la implementación del algoritmo proporcionara valores similares a los de la investigación en [20].

En la Tabla 3.1, se muestran los *datases* utilizados para hacer la comparativa de los tres algoritmos. En la primera columna, se muestra el nombre del *dataset*. En la segunda, *n* es el número de objetos. En la tercera, *d* es el número de dimensiones o atributos.

En la Tabla 3.2, se muestran los resultados de tiempo y calidad en la solución de los *datasets*. En la primera columna, se muestra el nombre del *dataset*. En la segunda, los *k* grupos que se formaron. En la tercera, cuarta y quinta, el porcentaje promedio de la reducción de tiempo

ρ (%). En las columnas sexta, séptima y octava el porcentaje promedio de reducción de calidad δ (%) de los algoritmos *O-K-Means* (*OKM*), *Fahim y OKM-Fahim* con respecto *al K-Means* estándar.

Tabla 3.1 *Datasets* utilizados para evaluar variantes secuenciales de *K-Means*

Dataset	***n***	***d***
Abalone	4177	7
Letters	20,000	16
Skin	245,057	3
EPCS	2,049,280	7

Tabla 3.2 Tiempo ρ (%) y calidad δ (%) de agrupamiento de *O-K-Means*, *Fahim* y *OKM-Fahim* respecto a *K-Means*

Dataset	***K***	% de disminución de tiempo, ***ρ* (%)**			% de disminución de calidad, ***δ* (%)**		
		OKM	***Fahim***	***OKM-Fahim***	***OKM***	***Fahim***	***OKM-Fahim***
Abalone	20	62.25	80.87	**85.95**	**-0.74**	-1.25	-1.60
	50	61.00	78.72	**86.40**	**-0.65**	-0.99	-1.44
Letter	50	75.92	73.74	**88.16**	-0.40	**-0.09**	-0.45
	100	74.47	78.55	**88.37**	-0.44	**-0.10**	-0.46
Skin	10	69.28	43.94	**79.18**	**0.14**	-0.85	-0.57
	25	61.51	42.16	**78.81**	-1.31	**0.09**	-2.96
EPCS	50	90.48	66.22	**95.47**	-1.14	**3.28**	0.54
	100	90.13	57.53	**95.08**	-0.15	0.48	**0.50**
Promedio		73.13	65.22	**87.18**	-0.59	**0.07**	-0.81
Desviación estándar		11.26	14.67	**5.81**	**0.45**	1.33	1.09

Nota: En negritas se resaltan los mejores resultados de disminución de tiempo y calidad

Con base en los resultados se eligió *O-K-Means*. Si bien *OKM-Fahim* fue la opción más rápida, también fue la que obtuvo mayor pérdida de calidad. Por otra parte, *Fahim* obtuvo el mejor promedio de pérdida de calidad, pero fue el más lento y tuvo mayor desviación estándar. Por otra parte, *O-K-Means* consiguió el segundo lugar tanto en la reducción de tiempo como en la disminución de calidad, además su mejor desempeño en tiempo fue con los *datasets* más grande y su desviación estándar muestran resultados constantes. Por estas razones *O-K-Means*, se seleccionó como el candidato ganador para ser paralelizado y distribuido. En el Anexo A: se muestra el algoritmo de *O-K-Means*, descrito a detalle en la tesis[20] o el artículo[6].

3.3 Algoritmo propuesto *HOK-Means*

El segundo paso de la metodología, fue desarrollar el algoritmo *HOK-Means*. El cual es un rediseño del algoritmo secuencial *O-K-Means*, ganador en el punto anterior. Para especificar *HOK-Means* fueron necesarios los siguientes pasos:

1. Identificar las etapas del algoritmo *O-K-Means* que se pueden ejecutar simultáneamente. Evidentemente la etapa de clasificación es la que se puede ejecutar simultáneamente. Debido a que en ella se calculan las distancias de cada objeto a los centroides, y se asignan al centroide más cercano. Asimismo, el registro de cambio de grupo con respecto a la iteración anterior, también se puede ejecutar simultáneamente. Estas tareas son las que más consumen tiempo de procesamiento, en cada iteración, además son tareas independientes, por lo tanto, es posible programarlas en paralelo, utilizando operaciones atómicas sin reducir la calidad del agrupamiento.

2. Asignación de piezas de trabajo concurrentes en múltiples procesos que se ejecutan en paralelo. Cada procesador realizará la clasificación y validación de convergencia de una porción del *dataset*, la cual corresponde al número de objetos entre el número de procesadores disponibles.

3. Distribuir los datos de entrada, salida e intermedios asociados con el programa. El *dataset* a resolver estará en cada nodo, de tal manera que cada procesador puede acceder a la porción que le corresponda. En cada iteración, se envía a cada procesador, el número de líneas a procesar, un identificador de procesador, el número de grupos a formar, el nombre del archivo, el número de atributos y objetos del *dataset*, el nombre del archivo de los centroides. A partir de la segunda iteración cada procesador almacena en disco un archivo correspondiente a las membresías de su porción de objetos. Cada procesador envía un arreglo de k renglones con la sumatoria de las distancias mínimas a cada centroide, y la cantidad de objetos que pertenecen a ese grupo; esta información es necesaria para recalcular los centroides en cada iteración. También cada procesador envía la sumatoria de los objetos que cambiaron de grupo; con esta información se evalúa la convergencia.

4. Gestión de accesos a datos compartidos por múltiples procesadores. Con el fin de evitar la concurrencia al archivo de membresías, cada procesador escribe y lee de un archivo diferente correspondiente a la porción de objetos que está procesando.

5. Sincronizar los procesadores en varias etapas de la ejecución del programa paralelo. Cuando termina cada procesador su tarea, se notifica y al terminar todos, se hace los cálculos para el recalculo de centroides y la validación de convergencia.

Para distribuir las tareas entre los procesadores del equipo, se utilizó una arquitectura maestro-esclavo. Cada esclavo, procesa de forma paralela una parte del trabajo, el cual se denomina tarea. Mientras, el maestro coordina el envío de tareas, recibe y calcula los resultados de todos los procesadores esclavos en cada iteración. Es decir, cada procesador esclavo hace uso de los recursos de su equipo. Al finalizar su tarea, envía el mensaje de término y sus resultados al maestro. La Figura 3.2, representa la arquitectura maestro esclavo, cada nodo (computadora), contiene *x* número de procesadores los cuales se denominan esclavos. Un solo procesador es maestro, el cual coordina la comunicación con cada procesador esclavo.

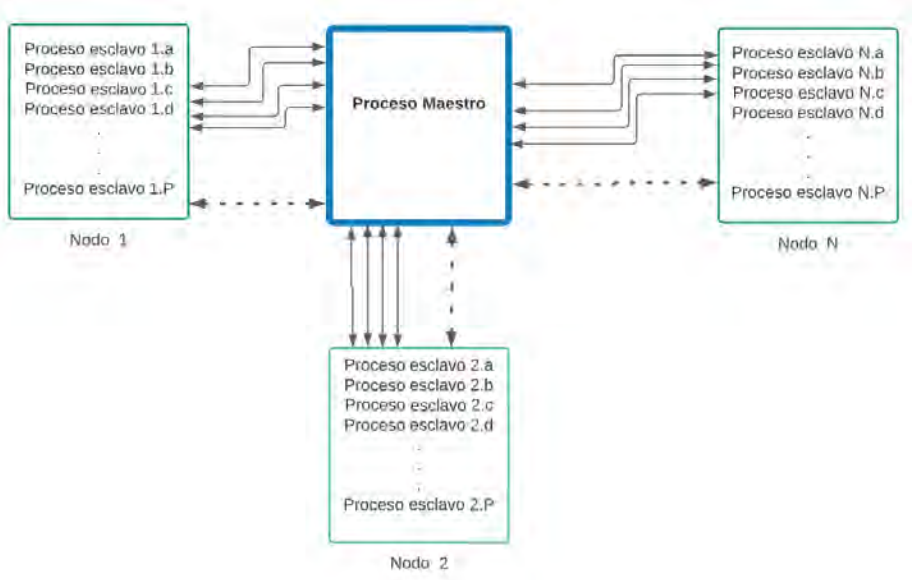


Figura 3.2 Arquitectura maestro-esclavo, utilizada con *HOK-Means*

Para describir el algoritmo *HOK-Means*, nos basaremos en el Algoritmo 1, el cual muestra la secuencia de ejecución de instrucciones en el procesador maestro y los procesadores esclavos.

Inicialización: La inicialización del procesador maestro se muestra en las líneas 2 a 7 del Algoritmo 1. En esta fase, se inicializan las variables. Un parámetro destacable es el valor del umbral; en este caso se asigna un valor de 0.72 U. En la línea 7, el procesador maestro envía la orden de inicio, los datos de los centroides y el identificador para los objetos en los que trabajará cada procesador esclavo.

Clasificación: Las líneas 8 a 15 muestran las instrucciones realizadas en paralelo por los procesadores esclavos. Las líneas 10 y 11 tienen las instrucciones para calcular la distancia de cada objeto a cada uno de los centroides. La línea 12 muestra la instrucción para asignar un objeto al grupo cuyo centroide está más cerca del objeto. La siguiente línea determina el número de objetos que cambiaron de grupo. Esta información es relevante para determinar cuándo debe detenerse el algoritmo. En la línea 14, los nodos esclavos transmiten al nodo maestro una matriz, cuyo número de filas corresponde al número de centroides. En la primera columna, está el valor del identificador del centroide; en la segunda, la suma de la distancia de todos los objetos que pertenecen a este grupo; y el tercero corresponde al número de objetos del grupo. La línea 15 transmite el número de objetos que cambiaron de grupo.

Cálculo de γ: La línea 17, muestra la instrucción donde el procesador maestro concentra los datos que le han transmitido todos los procesadores esclavos y calcula el porcentaje de cambio de objetos γ. Nótese que el valor de γ se utiliza en el criterio de parada del algoritmo.

Cálculo del centroide: En la línea 20 se realiza el cálculo de los nuevos centroides.

Convergencia: En la línea 22, se determina si el porcentaje de cambio de objeto es menor o igual al umbral predeterminado. Si es afirmativa, el algoritmo se detiene. De lo contrario, el algoritmo continúa en el paso 7.

Algoritmo 1 : *HOK-Means*

1 ***Procesador maestro***
2 **Inicialización:**
3 $P := \{p_1, \ldots, p_t\}$; // Carga el conjunto de procesadores disponibles
4 $N := \{x_1, \ldots, x_n\}$; // Carga el *dataset*
5 $M := \{\mu_1, \ldots, \mu_k\}$; // Inicializa los centroides
6 $U := 0.72$; //Valor del umbral para determinar la convergencia
7 Manda orden de inicio, centroides (M), y n_o a los esclavos;
8 ***Procesadores esclavos***
9 **Clasificación:**
10 For $x_i \in N_p$ and $\mu_k \in M$\{
11 Calcula la distancia Euclidiana de x_i a cada k centroide;
12 Asigna el objeto x_i al centroide más cercano μ_k;
13 Calcula el número de objetos que cambiaron de grupo;
14 Envía la matriz de resultados MR;
15 Envía el número de objetos que cambiaron de grupo $\boldsymbol{v_r}$
16 ***Procesador maestro***
17 Recibe el estado de fin e información de todos los nodos esclavos;
18 Cálcula el porcentaje de cambio $\gamma_r = 100(v_r/n)$;
19 **Cálculo de centroides:**
20 Calculate the centroid M;
21 **Convergencia:**
22 If ($\gamma \leq U$):
23 Detiene el algoritmo;
24 Otherwise:
25 Regresa al paso **7**;
26 Fin del algoritmo

En la Figura 3.3, se muestra el diagrama de flujo que describe el algoritmo paralelo *HOK-Means*, de lado izquierdo se observan el flujo para las tareas del proceso maestro y de lado derecho las tareas para los procesos esclavos, en cada una de las iteraciones. Las tareas del proceso maestro son ejecutadas de manera secuencial. Las tareas de los procesos esclavos, son ejecutados paralelamente y sobre la porción de objetos asignada.

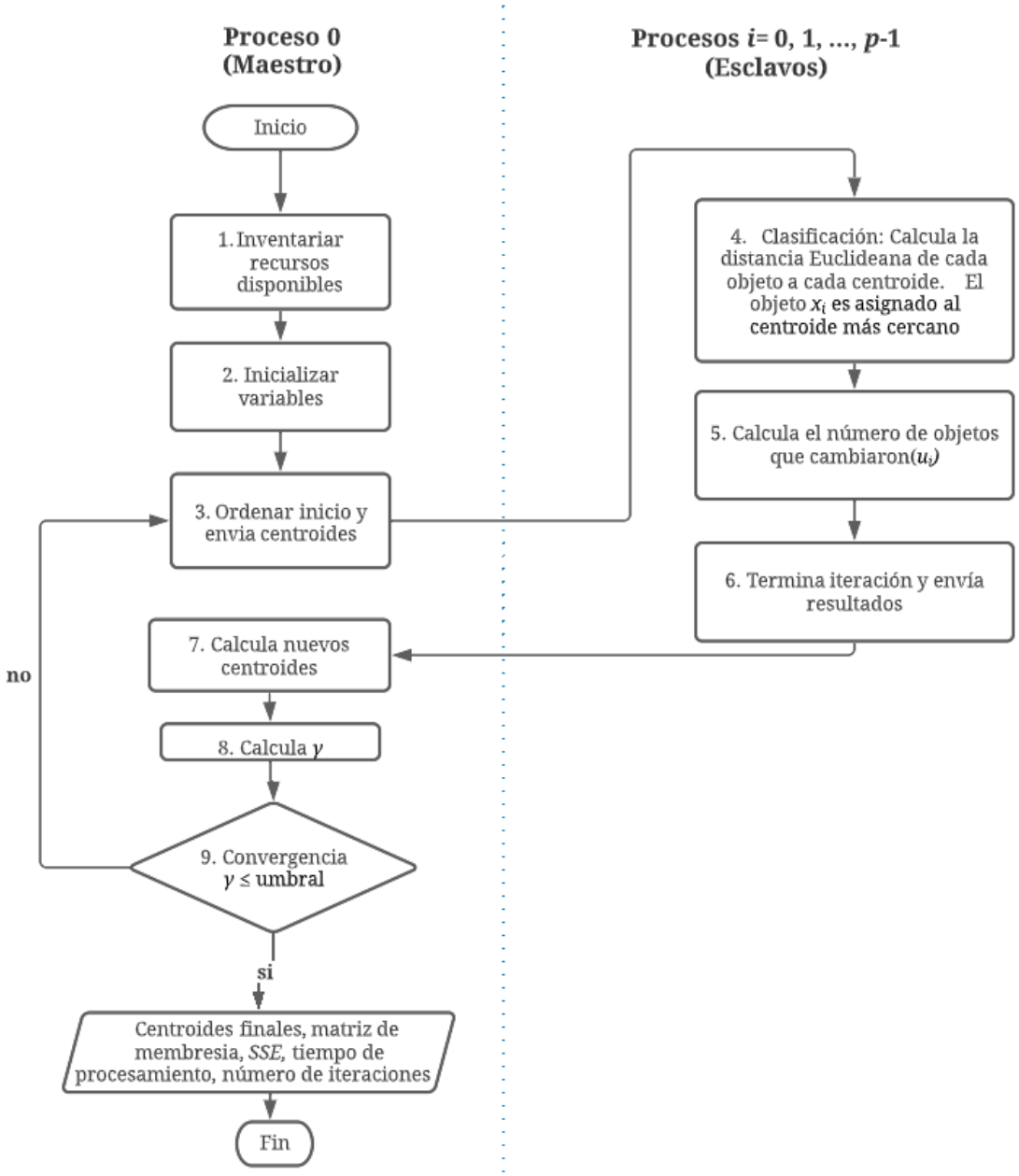


Figura 3.3 Diagrama de flujo de HOK-Means

3.3.1 Entorno de desarrollo

La programación del *HOK-Means*, se codificó en lenguaje *Python* 3.8, sobre un sistema operativo *Windows 10*. Aunque existen otros lenguajes y plataformas especializadas para la programación paralela, el propósito de esta tesis es evaluar el desempeño paralelo del rediseño del algoritmo *O-K-Means,* no evaluar las plataformas. Por lo cual se eligió el lenguaje *Python* aprovechando sus siguientes ventajas: es ampliamente utilizado en ciencia de datos y el *machine learning*, tiene una baja curva de aprendizaje, es un lenguaje intuitivo, posee una amplia variedad de librerías para el análisis de datos y procesamiento paralelo; además, cuenta con una amplia comunidad de desarrolladores y amplia documentación. Algunas librerías utilizadas fueron: *Pandas*, *Numpy* y *Parallel Python* (*pp*).

Las características de la librería *Parallel Python* y el diseño del algoritmo, hacen que *HOK-Means* tenga escalabilidad, es decir, la capacidad de ampliación de un sistema para satisfacer las necesidades empresariales o institucionales. Para escalar un sistema, debe agregar *hardware* adicional o actualizar el *hardware* existente sin modificar significativamente la aplicación. Es importante destacar, que *HOK-Means* es escalable en cuanto al número de nodos o equipos, ya que basta con agregar un equipo, en el cual se ejecute el módulo de *Parallel Python*, y el programa se encargará de agregar sus procesadores como esclavos. Cada procesador hace uso de los recursos de su equipo.

El diseño del algoritmo permite que el *dataset* sea distribuido entre el número de procesadores disponibles, cada procesador sabe con cuales objetos va realizar su tarea. Por lo cual en cada nodo se coloca el *dataset* completo. Una desventaja de hacerlo así es que el tamaño del *dataset* está restringido al espacio en disco, del nodo con menor capacidad. Sin embargo, evita el intercambio de información en red, o registro en disco.

Se implementó una arquitectura maestro-esclavo interconectando tres computadoras portátiles mediante un *router*, con una configuración de protocolo TCP/IP. Las características de los equipos utilizados son descritas en la Tabla 3.3 En cada nodo o laptop, se instalaron las mismas versiones de *Python* y librerías utilizadas.

Tabla 3.3 Infraestructura de *hardware* utilizada para experimentación paralela y distribuida de *HOK-Means*

Nodo	**Marca y modelo**	**No de procesadores y características**	**Memoria RAM**	**Disco**
1	Dell	8 Intel® Core™ i7-6700HQ CPU @ 2.60GHz	16GB	1TB
2 y 3	Asus Expertbook	8 11th Gen Intel® Core™ i7-1165G7 CPU 2.80 GHz	16G B	1TB
---	Router tp-link/TL-WR940N	Con 4 puertos LAN 4x10/100Mbps		

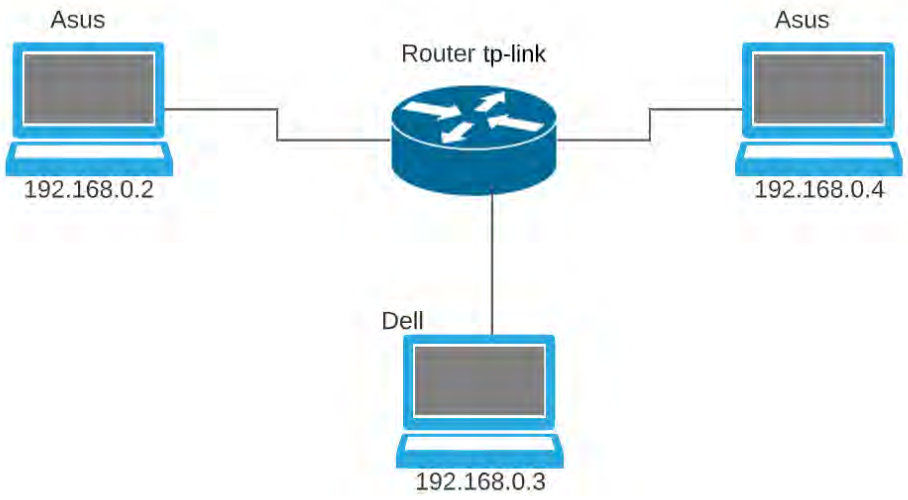


Figura 3.4 Configuración de red

3.3.2 Métricas utilizadas

Para evaluar el desempeño paralelo, se utilizaron la aceleración (*Sp*) y la eficiencia paralela (*Ep*), las cuales son dos métricas que permiten evaluar el rendimiento del procesamiento de un algoritmo paralelo.

La tasa de aceleración o *speedup*, indica la relación entre el tiempo de ejecución secuencial (*Ts*) y el tiempo de ejecución en paralelo (*Tp*) Eq. (1). El *speedup* ideal es un valor lineal *Sp* = *p*, donde *p* es el número de procesadores utilizados. La eficiencia paralela indica la relación entre *Sp* y el número de procesadores utilizados (*p*) Eq. (2). El valor ideal de *Ep* es uno.

$$\text{speedup} : Sp = \frac{Ts}{Tp} \tag{1}$$

$$\text{Eficiencia paralela} : Ep = \frac{Sp}{p} \tag{2}$$

Por otra parte, para medir la calidad del agrupamiento se utilizó el valor de la función objetivo, la suma del error cuadrático (SSE), según el algoritmo *K-Means*. ecuación (3) muestra SSE. La calidad del agrupamiento es superior cuando el valor de SSE es más bajo.

$$SSE = \sum_{j=1}^{k} \sum_{x \in \mu_j} \left\| x - \mu_j \right\|^2 \tag{3}$$

También se utilizó el porcentaje de pérdida de calidad (δ (%)) el cual es la relación de dos medidas de calidad $\delta(\%)=100(1-z/z_o)$, donde z es el SSE obtenido al resolver con *K-Means* estándar y z_o es el SSE obtenido por *O-K-Means*, que es igual al obtenido con *HOK-Means*.

Es conveniente mencionar que la complejidad computacional de *K-Means* es *O*(*ndk*) por cada iteración. En el resto del documento *ndk* se utiliza en la parte experimental como un indicador relacionado con el tamaño del *dataset*, el cual es producto de la cantidad de objetos en el conjunto de datos *n*, la cantidad de atributos *d* y la cantidad de grupos para formar *k.*

Capítulo 4

Validación experimental del algoritmo propuesto

Ciencia sin conciencia no es más que ruina del alma.

Francois Rabelais

El algoritmo *HOK-Means* fue validado de manera experimental dentro de un marco metodológico y se obtuvieron resultados alentadores. Con ese propósito, se diseñaron y ejecutaron un conjunto de experimentos con instancias tanto reales como sintéticas. En este capítulo se abarcan las fases 3, 4 y 5 del enfoque de solución relacionado con el diseño de experimento, análisis de resultados y análisis de las ventajas del algoritmo.

4.1 Descripción del entorno de pruebas

El *Benchmark* para llevar a cabo las pruebas experimentales, consta de *datasets* reales y sintéticos, las cuales se solucionaron con los algoritmos *K-Means*, *O-K-Means* y *HOK-Means*.

Los *datasets* reales se obtuvieron del reconocido repositorio de la Universidad de California, UCI Machine Learning Repository [21]. Los *datasets* sintéticos utilizados, fueron los que se crearon y utilizaron en la tesis que dio origen al algoritmo base *O-K-Means* [20]. En el Experimento III, se utilizaron los dos *datasets* más grandes reales y sintéticos del Experimento II, los cuales además fueron escalados en 10 y 50 veces de forma horizontal o vertical.

En la selección y generación de los *datasets*, se tomó en cuenta que tanto su tamaño como sus características fueran como los que se presentan en el paradigma de *Big Data* [47]. De manera general, se utilizaron *datasets* con valores grandes de n y d, lo que permitió estudiar y validar el rendimiento del algoritmo para valores altos del indicador *ndk*. En promedio, los *datasets* utilizados se caracterizaron por ser de un tamaño superior a un millón de objetos con hasta 70 dimensiones.

Es conveniente destacar que en la Sección 4.2 se describen los casos de prueba de los tres experimentos y en las Secciones 4.3, 4.4 y 4.5 se muestran los resultados de aplicar los algoritmos indicados en cada experimento.

4.2 Diseño de experimentos

Con el fin de evaluar el desempeño de *HOK-Mean*s se diseñaron tres experimentos con los siguientes objetivos:

- Experimento I. - Comprobar el correcto funcionamiento del *HOK-Means.*

- Experimento II. – Utilizando un solo equipo, comparar el tiempo de solución de *datasets*, utilizando *O-K-Means* y *HOK-Means*, tanto con:
    a) *datasets* reales,
    b) *datasets* sintéticos.

- Experimento III. – Utilizando más de un equipo, comparar la robustes y el tiempo de solución con *datasets* grandes y escalados, usando *O-K-Means* y *HOK-Means*.

En lo que sigue denotamos t, t_o, t_p, como el tiempo promedio de solución de *K-Means*, *O-K-Means* y *HOK-Means*, respectivamente. El valor z es el promedio de la función objetivo de *K-Means*; $z_{o=p}$ es el valor promedio de la función objetivo tanto de *O-K-Means* y *HOK-Means*, para ambas es igual. El resultado de la solución z de cada *dataset* está dado por la suma del error al cuadrado, *SSE*.

4.3 Resultados del Experimento I

Con el objetivo de validar que los resultados que se obtienen al ejecutar el algoritmo *HOK-Means* sean correctos, se resolvieron tres *datasets* reales reconocidos, utilizando *O-K-Means* y *HOK-Means*. Cada *dataset* se resolvió usando los mismos valores de entrada: el valor de k y los valores de los centroides iniciales en cada ejecución. Consecuentemente, se validó que los valores de salida, los cuales son: el número de iteraciones, la pertenencia de los objetos, los centroides finales y *SSE*, tanto del algoritmo *O-K-Means* y *HOK-Means* resultaran iguales.

La Tabla 4.1 contiene la descripción de los *datasets* utilizados en este experimento. En la primera columna, el nombre del *dataset*. En la segunda, n es el número de objetos. En la tercera, d es el número de atributos.

En la Tabla 4.2, se presentan los resultados de las soluciones. Cada renglón representa el promedio de 30 ejecuciones de la solución del *dataset* dada una k, tanto para *O-K-Means* como para *HOK-Means*. En la primera columna, se muestra el nombre de *dataset*. En la segunda, k es el número de grupos que se formaron. En la tercera, se muestra el valor de la calidad de la solución obtenida del algoritmo *O-K-Means* secuencial (z_o), la cual es igual a la de *HOK-Means* paralelo (z_p), ver cuarta columna. En la quinta, se muestra el número de iteraciones requeridas en la solución con *O-K-Means* (I_o), las cuales son igual que con *HOK-Means* (I_p), ver sexta columna. La séptima y octava columna muestran el tiempo de solución en segundos del algoritmo *O-K-Means*

secuencial (t_o) y *HOK-Means* (t_p).

En todos los casos tanto los archivos de salida: los centroides finales y las membresías de los objetos, obtenidos por cada algoritmo, fueron iguales. Además, que como se observa en la Tabla 4.2 los resultados de la calidad del agrupamiento y el número de iteraciones para ambos algoritmos son iguales. Utilizando *HOK-Means*, se obtuvieron las mismas salidas que el *O-K-Means*, en un menor tiempo. Con este experimento se comprueba que el algoritmo de *HOK-Means* se programó correctamente.

Tabla 4.1 *Datasets* utilizados en el Experimento I y sus características

Dataset	***n***	***d***
Letters	20,000	16
Skin	245,057	3
EPCS	2,049,280	7

Tabla 4.2 Resultados del Experimento I

Dataset	***k***	z_o	z_p	I_o	I_p	t_o	t_p
Letters	50	494,237.1	494,237.1	23	23	11.2	11.1
	100	370,272.9	370,272.9	18	18	18.0	13.2
Skin	50	71,626,703.6	71,626,703.6	29	29	31.4	27.4
	100	36,836,533.2	36,836,533.2	26	26	58.4	34.6
EPCS	50	15,161,560.0	15,161,560.0	47	47	1,025.9	319.8
	100	11,993,364.6	11,993,364.6	51	51	2,249.0	635.9

4.4 Resultados del Experimento II con *datasets* sintéticos

En este experimento se realizó el agrupamiento de 15 diferentes *datasets* sintéticos, cada uno con uno o más valores de *k*. Cada *dataset* se solucionó con los algoritmos *O-K-Means* y *HOK-Means*. En cada prueba se controló que los centroides iniciales fueran iguales. Se realizó una muestra de 30 ejecuciones para cada prueba. En total se solucionaron 1,380 *datasets* dado una *k* y un archivo de centroides iniciales con los dos algoritmos. Asimismo, para tener una comparativa real con el

K-Means estándar, se realizó una correlación entre promedio del número de iteraciones (*I*) del algoritmo y el tiempo promedio necesario para realizar una iteración con el algoritmo *K-Means.* La experimentación los códigos de los algoritmos *K-Means* y *O-K-Means* se programaron en Lenguaje C, con un compilador GCC 4.9.2.

En la Tabla 4.3, se describe el conjunto de *datasets* sintéticos utilizados. En la primera fila, se indica el identificador del *dataset*. En la segunda, el número de objetos (los valores de *n* están en millones). En la tercera, el número de atributos o dimensiones *d*.

Tabla 4.3 *Datasets* sintéticos utilizados en el Experimento II

Id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n	2	2	2	2	2	1	1	1	1	1	0.5	0.5	0.25	1	1.2
d	2	4	6	5	7	2	3	4	7	10	2	11	12	11	5

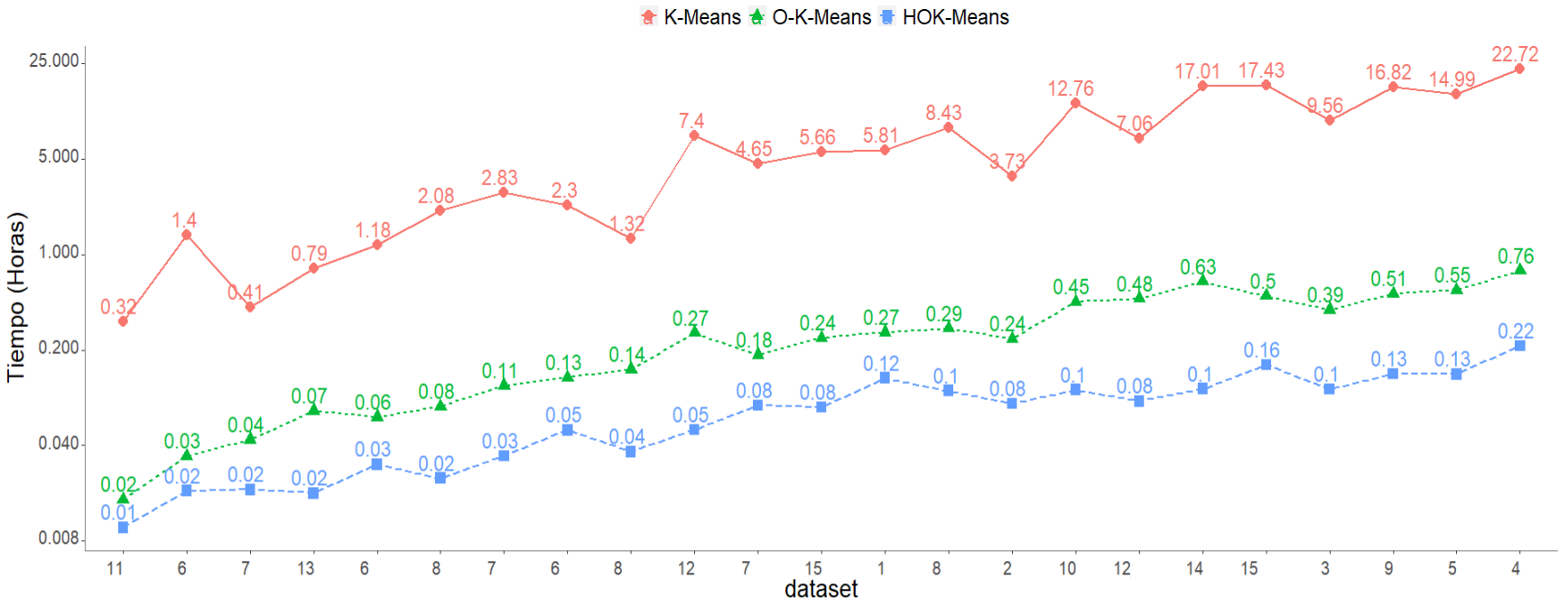
En la Tabla 4.4, cada renglón representa el promedio de los resultados de 30 ejecuciones de la solución de cada *dataset* sintético dada una *k*. La tabla esta ordenada por el indicador de tamaño *ndk*. En la primera columna, se muestra el identificador de la prueba. En la segunda, el Id del *dataset*, el cual se relaciona con el Id de la Tabla 4.3. En la tercera, *k* es el número de grupos que se formaron. En la cuarta, el indicador *ndk* dado en millones. En las columnas quinta, sexta y séptima muestran el tiempo de ejecución en horas de los algoritmos *K-Means* estándar ($t_{K\text{-}Means}$), *O-K-Means* secuencial ($t_{O\text{-}K\text{-}Means}$) y *HOK-Means* ($t_{HOK\text{-}Means}$), respectivamente. En la octava y novena muestran la calidad del agrupamiento con el *K-Means* estándar (*z*) y el *O-K-Means* ($z_{O\text{-}K\text{-}Means}$), el cual fue el mismo para el *HOK-Means*.

En la gráfica de la Figura 4.1, se observan los tiempos de procesamiento, al solucionar *datasets* sintéticos, de la Tabla 4.4, obtenidos con *K-Means*, *O-K-Means* y *HOK-Means*. Los resultados se ordenaron de manera ascendente por el indicador *ndk*. En el eje horizontal, se muestra el identificador del *dataset*, el cual corresponde con el *dataset* en la Tabla 4.3. En la parte inferior de la Figura 4.1, se muestra en la primera fila, el indicador *ndk* dado en millones. En la segunda, el número de grupos que se formaron *k*. En la tercera, el número de atributos *d*. En la cuarta, el número de objetos *n*. En la última, el porcentaje de pérdida de calidad de *HOK-Means* con respecto a *K-Means* δ. Es destacable que, en cada caso el tiempo de procesamiento con *HOK-Means* es el menor. Por ejemplo, para el *dataset* 14 el tiempo necesario para resolverlo con *K-Means* fue de

17.01 horas, mientras que utilizando *O-K-Means* en 36 minutos se pudo resolver y *con HOK-Means* fueron suficientes 6 minutos.

Tabla 4.4 Tiempo y calidad de solución de *datasets* sintéticos con *K-Means, O-K-Means* y *HOK-Means*

Id.	***dataset***	***k***	***ndk***	$t_{K\text{-}Means}$	$t_{O\text{-}K\text{-}Means}$	$t_{HOK\text{-}Means}$	z	$z_{O\text{-}K\text{-}Means}$
1	11	50	50	0.32	0.02	0.01	26,959	27,131
2	6	50	100	1.40	0.03	0.02	53,988	54,374
3	7	50	150	0.41	0.04	0.02	130,439	130,969
4	13	50	150	0.79	0.07	0.02	188,184	188,518
5	6	100	200	1.18	0.06	0.03	38,065	38,345
6	8	50	200	2.08	0.08	0.02	211,690	212,958
7	7	100	300	2.83	0.11	0.03	103,010	103,444
8	6	200	400	2.30	0.13	0.05	26,848	27,047
9	8	100	400	1.32	0.14	0.04	177,224	177,876
10	12	100	550	7.40	0.27	0.05	324,023	324,771
11	7	200	600	4.65	0.18	0.08	81,418	81,817
12	15	100	600	5.66	0.24	0.08	28,606,830	28,675,428
13	1	200	800	5.81	0.27	0.12	53,722	54,149
14	8	200	800	8.43	0.29	0.10	147,867	148,482
15	2	100	800	3.73	0.24	0.08	354,738	355,811
16	10	100	1,000	12.76	0.45	0.10	588,590	589,829
17	12	200	1,100	7.06	0.48	0.08	299,454	300,141
18	**14**	**100**	**1,100**	**17.01**	**0.63**	**0.10**	**648,210**	**649,542**
19	15	200	1,200	17.43	0.50	0.16	25,001,825	25,090,193
20	3	100	1,200	9.56	0.39	0.10	648,108	650,450
21	9	200	1,400	16.82	0.51	0.13	353,458	354,372
22	5	100	1,400	14.99	0.55	0.13	776,098	778,426
23	4	200	2,000	22.72	0.76	0.22	434,530	435,633



ndk	50	100	150	150	200	200	300	400	400	550	600	600	800	800	800	1,000	1,100	1,100	1,200	1,200	1,400	1,400	2,000
k	50	50	50	50	100	50	100	200	100	100	200	100	200	200	100	100	200	100	200	100	200	100	200
d	2	2	3	12	2	4	3	2	4	11	3	5	2	4	4	10	11	11	5	6	7	7	5
n	0.5	1	1	0.25	1	1	1	1	1	0.5	1	1.2	2	1	2	1	0.5	1	1.2	2	1	2	2
δ (%)	-0.63	-0.71	-0.4	-0.17	-0.73	-0.59	-0.42	-0.74	-0.36	-0.23	-0.49	-0.23	-0.79	-0.41	-0.3	-0.21	-0.22	-0.2	-0.35	-0.36	-0.25	-0.29	-0.25

Figura 4.1 Tiempo de solución de *datasets* sintéticos con *K-Means*, *O-K-Means* y *HOK-Means*.

En la Tabla 4.5, se presentan los porcentajes de reducción de tiempo y calidad que se lograron con *HOK-Means* con respecto a *K-Means* y *O-K-Means*, relacionados con los experimentos descritos en la Tabla 4.4. En la primera columna, se muestra el identificador del experimento. En la segunda, el porcentaje de reducción de tiempo de *HOK-Means* con respecto a *O-K-Means* $\rho_{pd} = 100\left(1 - t_o / t_p\right)$. En la tercera, el porcentaje de reducción de tiempo de *HOK-Means* con respecto a *K-Means* $\rho_t = 100\left(1 - t / t_p\right)$. En la cuarta, el porcentaje de reducción de calidad de *HOK-Means* con respecto a *K-means* $\delta_t = 100\left(1 - z / z_{o=p}\right)$. En la quinta, el valor del *speedup* de *HOK-Means* con respecto a *O-K-Means* $Sp = t_o / t_p$. En la sexta, el valor de la eficiencia paralela $Ep = Sp / \#\ procesadores$, donde #procesadores es el número de procesadores utilizados para la ejecucion paralela. Para este experimento, empleando un nodo, el número de procesadores es ocho.

En la gráfica de la Figura 4.2, se observa el desempeño del *speedup* correspondiente a cada uno de los *datasets* solucionados. Se puede observar que existe una correlacion entre el desempeño del *speedup* con los valores del indicador *ndk* y el número de atributos. Es decir, con *datasets* cuyo valor del indicador *ndk* sea igual, el *speedup* es mas alto con el de mayor número de atributos. En el mejor de los casos, con el *dataset* 14, el tiempo de solución se redujo de 17 horas a menos de seis minutos, es decir 99.4% del tiempo de procesamiento respecto de *K-Means*, con pérdida de calidad de solamente 0.2%, logrando un *Sp* de 6.1 y una *Ep* de 0.76 con respecto a *O-K-Means*.

Tabla 4.5 Porcentajes de reducción de tiempo, calidad, S*p* y *Ep* obtenidos en la solución de *datasets* sintéticos.

Id	ρ_{pd} (%)	ρ_t (%)	δ_t (%)	*Sp*	*Ep*
1	**37.6**	96.9	-0.6	**1.6**	0.20
2	44.1	98.7	-0.7	1.8	0.22
3	56.9	**95.4**	-0.4	2.3	0.29
4	75.2	97.7	-0.2	4.0	0.50
5	55.1	97.5	-0.7	2.2	0.28
6	70.1	98.9	-0.6	3.3	0.42
7	69.7	98.8	-0.4	3.3	0.41
8	58.7	97.8	-0.7	2.4	0.30
9	75.2	97.3	-0.4	4.0	0.50
10	80.4	99.3	-0.2	5.1	0.64
11	57.6	98.3	-0.5	2.4	0.29
12	68.6	98.6	-0.2	3.2	0.40
13	53.8	97.9	-0.8	2.2	0.27
14	65.0	98.8	-0.4	2.9	0.36
15	66.5	97.8	-0.3	3.0	0.37
16	77.2	99.2	-0.2	4.4	0.55
17	82.3	98.8	-0.2	5.7	0.71
18	**83.6**	**99.4**	-0.2	**6.1**	**0.76**
19	68.6	99.1	-0.4	3.2	0.40
20	73.8	98.9	-0.4	3.8	0.48
21	74.0	99.2	-0.3	3.8	0.48
22	75.7	99.1	-0.3	4.1	0.51
23	71.7	99.1	-0.3	3.5	0.44

Se resalta en negritas los mejores y peores resultados.

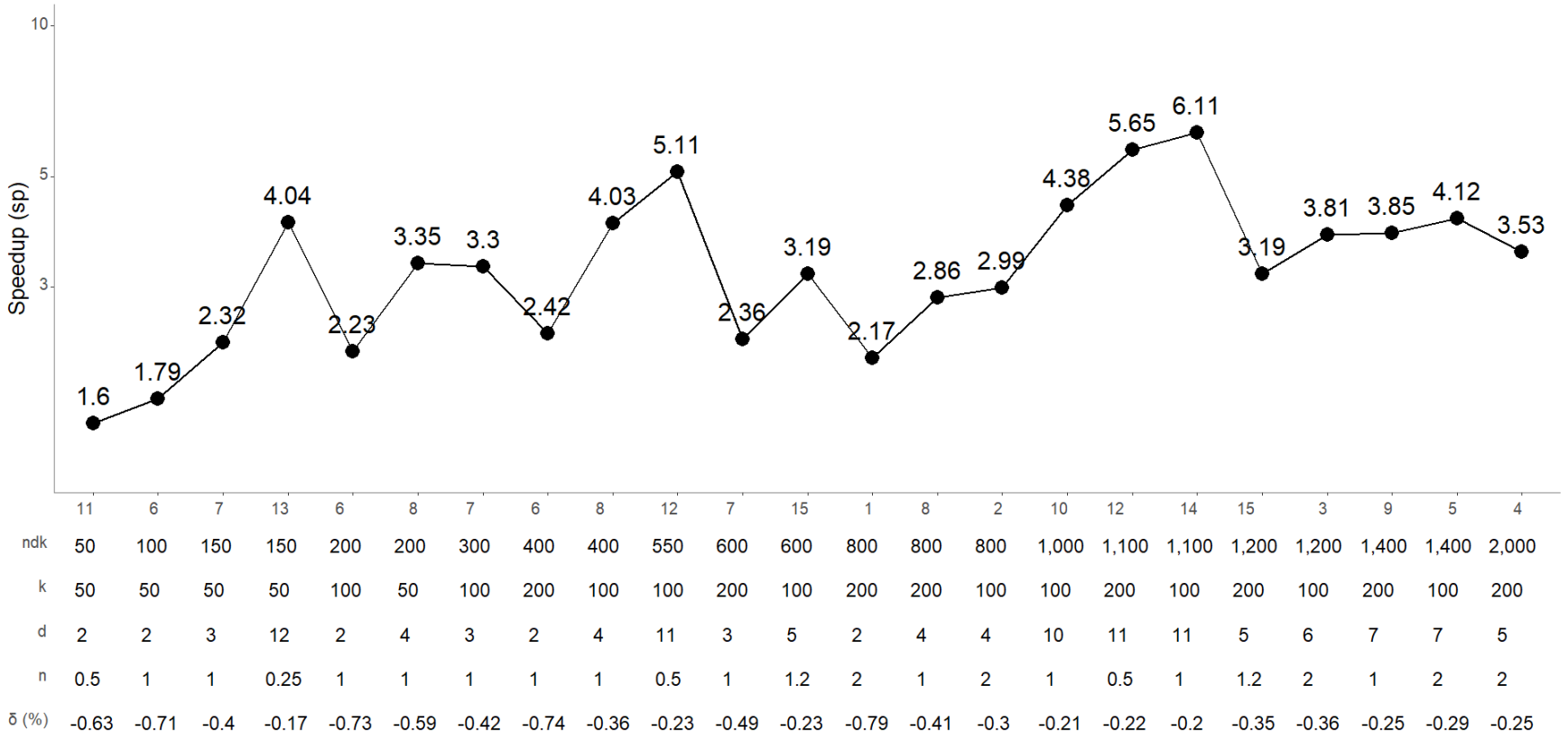


Figura 4.2 *Speedup* obtenido al resolver *datasets* sintéticos con *HOK-Means* vs *O-K-Means*

4.5 Resultados del Experimento II con *datasets* reales

Se realizó el agrupamiento de cinco diferentes *datasets* reales, utilizando dos valores de *k*. Para cada uno, se ejecutaron los algoritmos *K-Means* estándar, *O-K-Means* y *HOK-Means*. Se realizaron 30 ejecuciones con diferentes centroides iniciales en cada ejecución. En total se solucionaron 600 *datasets* reales dado una *k* y un archivo de centroides iniciales con los dos algoritmos. Asimismo, para tener una comparativa real con el *K-Means* estándar, se realizó una correlación entre promedio del número de iteraciones (*I*) del algoritmo y el tiempo promedio necesario para realizar una iteración con el algoritmo *K-Means* en Lenguaje C, en nuestros equipos.

En la Tabla 4.6, se describe el conjunto de *datasets* reales[21]. En el primer renglón, se indica el nombre del *dataset*. En el segundo, el número de objetos. En el tercero, el número de atributos.

Tabla 4.6 Características de *datasets* reales utilizados en el Experimento II

Dataset	*Abalone*	*Wind*	*Letters*	*DSAS*	*EPCS*
n	4,177	6,574	20,000	1,140,000	2,075,259
d	7	15	16	45	7

En la Tabla 4.7, cada renglón presenta los promedios de los resultados de 30 ejecuciones de los *datasets* reales pequeños. En la primera columna, se muestra el identificador del experimento. En la segunda, el nombre del *dataset*. En la tercera el número de grupos que se formaron. En la cuarta, el indicador *ndk*. En la quinta, sexta y séptima, se muestran el tiempo de ejecución en segundos, de los algoritmos *K-Means* estándar (t), *O-K-Means* secuencial (t_o) y *HOK-Means* (t_p), respectivamente. En la octava y novena, se muestran la calidad del agrupamiento con el *K-Means* estándar (z) y el *O-K-Means*, el cual es el mismo para el *HOK-Means*. En la Tabla 4.8, se muestran los resultados de la solución de *datasets* reales grandes, esta contiene la misma información que la Tabla 4.7, con la diferencia de que las columnas quinta, sexta y séptima tienen como unidad la hora en lugar de segundos.

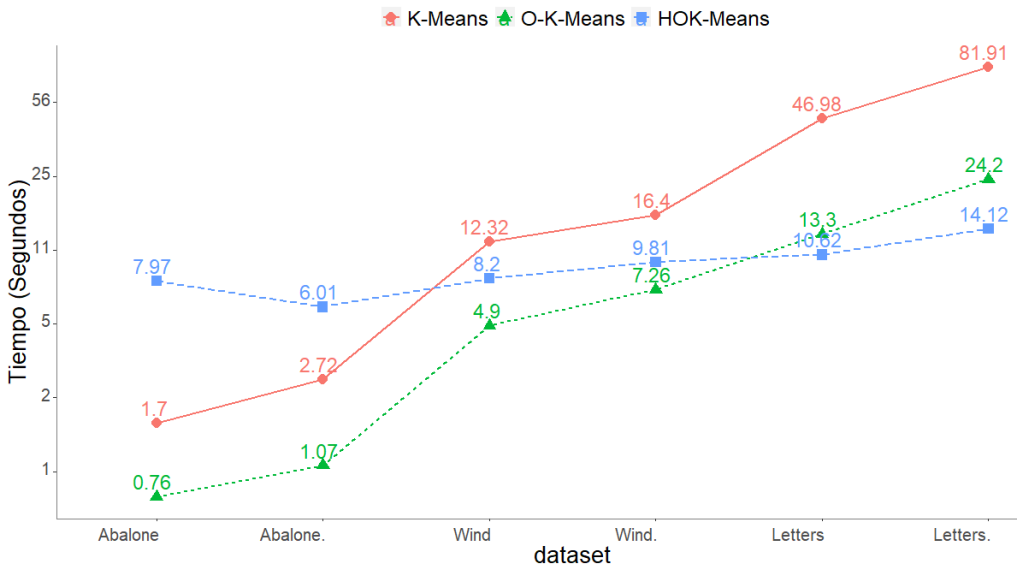
Tabla 4.7 Resultados de tiempo y calidad al resolver los *datasets* reales pequeños con *K-Means, O-K-Means* y *HOK-Means*

Id.	*dataset*	*k*	*ndk*	$t_{K\text{-}Means}$	$t_{O\text{-}K\text{-}Means}$	$t_{HOK\text{-}Means}$	z	$z_{O\text{-}K\text{-}Means}$
1	*Abalone*	20	0.6	1.70	0.76	7.97	314	317
2		50	1.5	2.72	1.07	6.01	247	248
3	*Wind*	80	7.9	12.32	4.90	8.20	57,657	57,707
4		160	15.8	16.40	7.26	9.81	52,598	52,697
5	*Letters*	50	16.0	46.98	13.30	10.62	95,050	95,271
6		100	32.0	81.91	24.20	14.12	82,330	82,476

Tabla 4.8 Resultados de tiempo y calidad al resolver los *datasets* reales grandes con *K-Means, O-K-Means y HOK-Means*

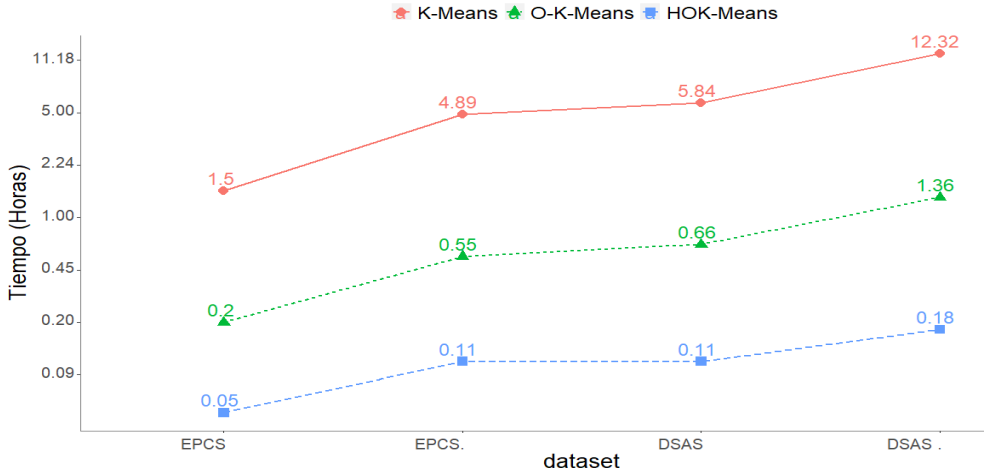
Id.	*dataset*	*k*	*ndk*	$t_{K\text{-}Means}$	$t_{O\text{-}K\text{-}Means}$	$t_{HOK\text{-}Means}$	z	$z_{O\text{-}K\text{-}Means}$
7	*EPCS*	50	717	1.50	0.20	0.05	3,299,098	3,285,982
8		100	1,434	4.89	0.55	0.11	2,569,076	2,574,634
9	*DSAS*	50	2,565	5.84	0.66	0.11	10,112,417	10,144,623
10		100	5,130	12.32	1.36	0.18	8,735,706	8,761,274

En la Figura 4.4, se muestra que en todos los experimentos con *datasets* reales grandes los tiempos de procesamiento de la propuesta *HOK-Means*, marcados de color azul, son los menores. En este caso, el mejor resultado se obtuvo con el *dataset DSAS* con un valor de *k* igual a 100, con la cual *HOK-Means* redujo en 98.5% el tiempo de procesamiento con respecto a *K-Means* con una pérdida en la calidad de la solución de sólo el 0.29%, logrando un *speedup* del 7.54 con respecto a *O-K-Means*.



ndk	0.58	1.46	7.89	15.78	16	32
k	20	50	80	160	50	100
d	7	7	15	15	16	16
n	4,177	4,177	6,574	6,574	20000	20000
δ (%)	-0.96	-2.23	-0.12	-0.18	-11.71	-0.75

Figura 4.3 Tiempo de solución de *datasets* reales pequeños con *K-Means, O-K-Means* y *HOK-Means*



ndk	717.2	1,434.50	2,565.00	5,130.00
k	50	100	50	100
d	7	7	45	45
n	2,049,280	2,049,280	1,140,000	1,140,000
δ (%)	0.4	-0.22	-0.32	-0.29

Figura 4.4 Tiempo de solución de *datasets* reales grandes con *K-Means*, *O-K-Means* y *HOK-Means*.

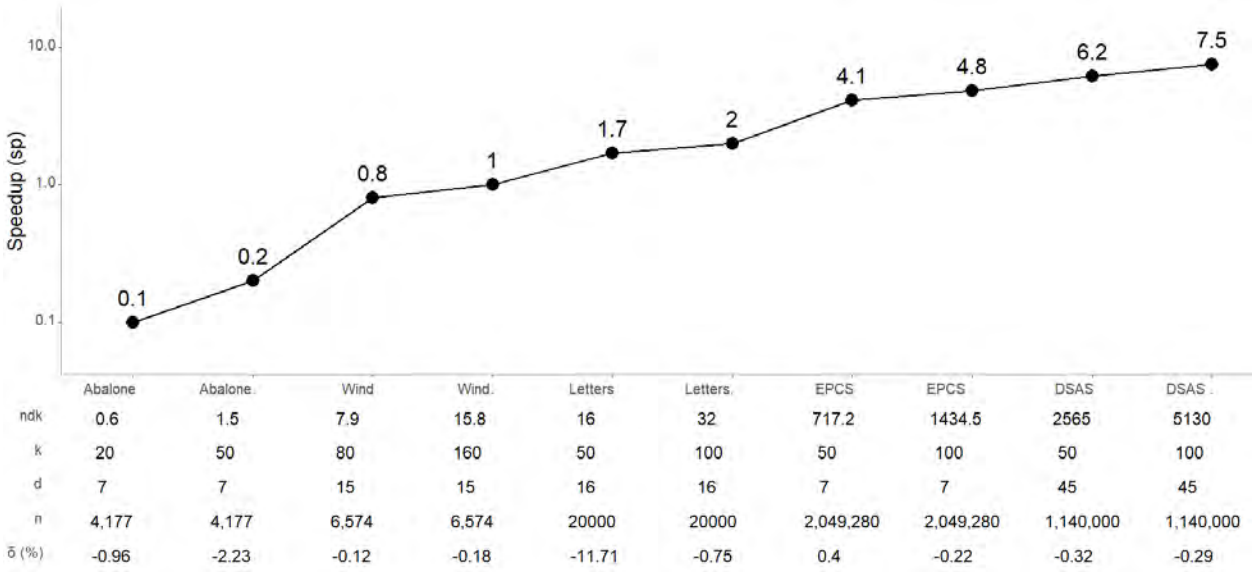
En la Tabla 4.9, se presentan las mismas columnas que en la Tabla 4.4, correspondiente a los resultados de la experimentación con los *datasets* reales, descritos en la Tabla 4.5. Al igual

que en la literatura, debido principalmente que la sincronización de procesos paralelos tiene un costo de sincronización [48], en *datasets* pequeños el paralelismo aumenta el tiempo de ejecución. Aunque en los dos *datasets* más pequeños (*Abalone* y *Wind*) no hubo reducción de tiempo, el tiempo necesario para resolverlos es de solamente 16 segundos y con *O-K-Means* de únicamente 9.81 segundos, véase Figura 4.3.

Tabla 4.9 Porcentajes de reducción de tiempo, calidad, *Sp* y *Ep* obtenidos en la solución de *datasets* reales

dataset	Id	ρ_{pd} (%)	ρ_t (%)	δ_t (%)	*Sp*	*Ep*
pequeños	1	-952.3	-367.5	-0.96	0.1	0.01
	2	-462.9	-120.9	-0.58	0.2	0.03
	3	-67.1	33.5	-0.09	0.8	0.09
	4	-35.1	40.2	-0.19	1.0	0.13
	5	20.2	77.4	-0.23	1.7	0.22
	6	41.6	82.8	-0.18	2.0	0.25
Grandes	7	75.4	96.7	0.40	4.07	0.51
	8	79.3	97.7	-0.22	4.83	0.60
	9	83.8	98.2	-0.32	6.16	0.77
	10	86.7	**98.5**	**-0.29**	**7.54**	0.94

La Figura 4.5, muestra el *speedup* obtenido al procesar los *datasets* reales, ordenado por el valor de *ndk.* Para los *datasets* reales se observa una relación directa de este índice con el *speedup*. El mejor resultado se obtuvo con el *dataset DSAS* y k=10, utilizando los *datasets* con el mayor indicador *ndk*, con el cual se obtuvo un *speedup* de 7.54, es decir, que utilizando el mismo equipo y *HOK-Means,* en lugar de 12.32 horas en solo 11 minutos se realizó el agrupamiento.



	Abalone	Abalone.	Wind	Wind.	Letters	Letters.	EPCS	EPCS	DSAS	DSAS.
ndk	0.6	1.5	7.9	15.8	16	32	717.2	1434.5	2565	5130
k	20	50	80	160	50	100	50	100	50	100
d	7	7	15	15	16	16	7	7	45	45
n	4,177	4,177	6,574	6,574	20000	20000	2,049,280	2,049,280	1,140,000	1,140,000
δ̄ (%)	-0.96	-2.23	-0.12	-0.18	-11.71	-0.75	0.4	-0.22	-0.32	-0.29

Figura 4.5 *Speedup* obtenido al resolver *datasets* reales con *HOK-Means* vs *O-K-Means*

4.6 Resultados del Experimento III

Con el fin de comparar el desempeño del *HOK-Means* con más de un nodo (equipo) y con *datasets* más grandes, se resolvieron los cuatro *datasets* más grandes del Experimento II, dos sintéticos (2ms7d, dino) y dos reales (*EPCS*, *DSAS*). Además, con el fin de generar *datasets* más grandes, estos se escalaron, tanto en número de dimensiones como en número de objetos. Adicionalmente se hicieron agrupaciones con valores de *k* igual a 50, 100 y 200. Cabe destacar que estos *datasets* escalados, por su tamaño ya no pudieron ser resueltos en nuestros equipos con el algoritmo secuencial *O-K-Means*, el cual fue la base de nuestra propuesta *HOK-Means*.

En la Tabla 4.10, se describe los *datasets* utilizados para el Experimento III. La primera columna, indica el nombre del *dataset*. La segunda, el número de objetos. La tercera, el número de atributos.

En la Tabla 4.11, cada renglón presenta los promedios de los resultados de 16 ejecuciones, para solucionar cada *dataset* dada una *k*. Se utilizaron uno, dos y tres nodos. En la primera

columna, se muestra el identificador del experimento. En la segunda, el nombre del *dataset*. En la tercera, el número de grupos que se formaron. En la cuarta, el indicador *ndk*. En las columnas quinta, sexta y séptima muestran el tiempo de ejecución en horas del algoritmo *HOK-Means* utilizando un nodo (t_{p_1n}), dos nodos (t_{p_2n}) y tres nodos (t_{p_3n}). En la octava, se muestra la calidad de agrupamiento (z_{*nodos}), la cual es la misma para uno, dos o tres nodos e igual que el *O-K-Means* secuencial.

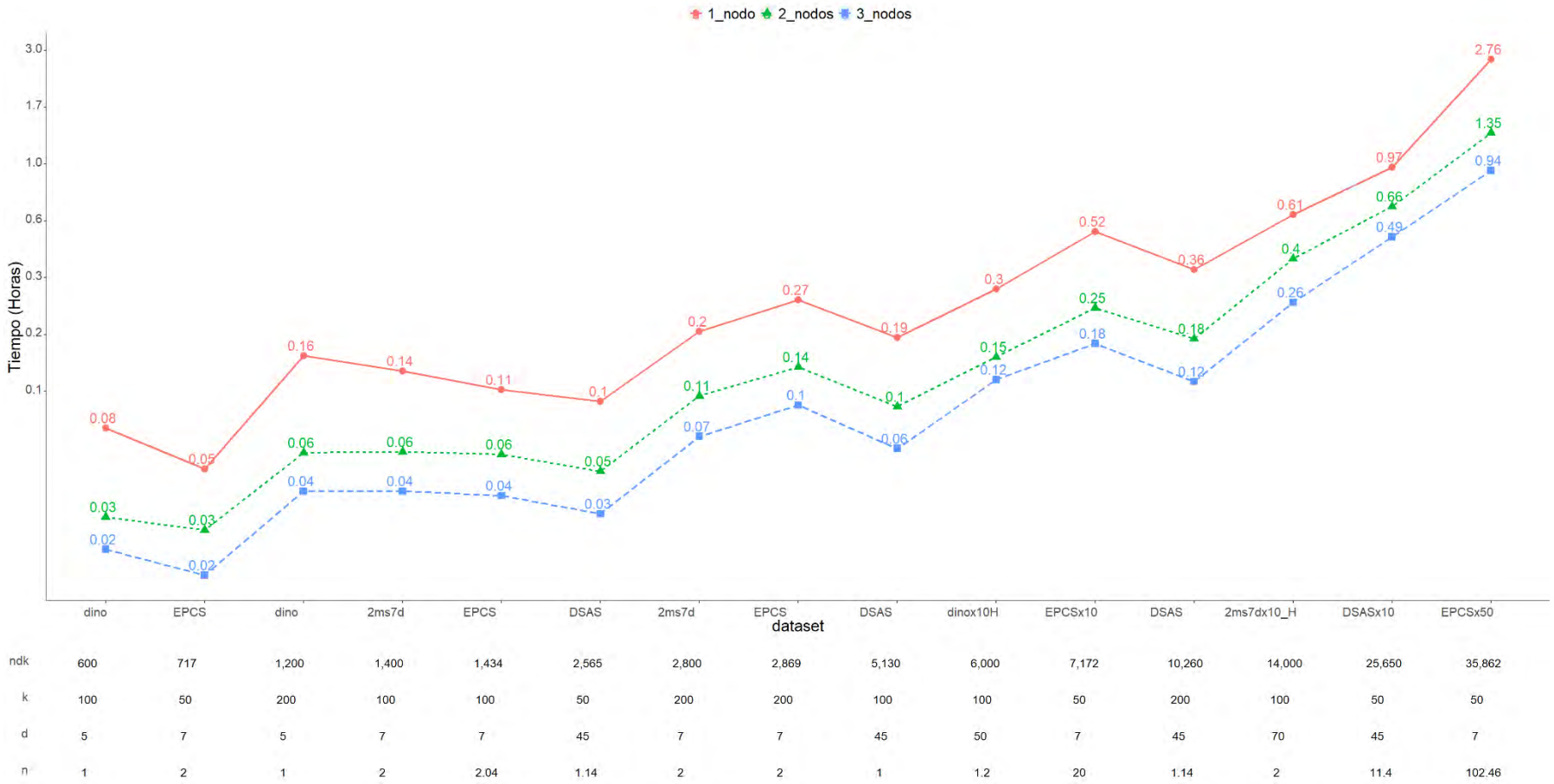
Tabla 4.10 Características de *datasets* utilizados en el Experimento III

Id	***n***	***d***
2ms7d	2,000,000	7
2ms7dx10_H	2,000,000	70
dino	1,200,000	5
dinox10H	1,200,000	50
DSAS	1,140,000	45
DSASx10	11,400,000	45
EPCS	2,049,280	7
EPCSx10	20,492,800	7
EPCSx50	102,464,000	7

Tabla 4.11 Resultados de tiempo y calidad al resolver *datasets* grandes con *HOK-Means* utilizando uno, dos y tres nodos

Id.	*dataset*	*k*	*ndk*	t_{p_1n}	t_{p_2n}	t_{p_3n}	z^{*}_{nodos}
1	dino	100	600	0.078	0.033	0.024	28,675,604
2	EPCS	50	717	0.052	0.027	0.019	3,291,226
3	dino	200	1,200	0.157	0.061	0.042	25,089,131
4	2ms7d	100	1,400	0.135	0.062	0.042	778,324
5	EPCS	100	1,434	0.113	0.060	0.041	2,565,515
6	DSAS	50	2,565	0.101	0.051	0.034	10,125,533
7	DSAS	100	5,130	0.187	0.096	0.064	8,779,293
8	2ms7d	200	2,800	0.198	0.106	0.072	709,044
9	EPCS	200	2,869	0.269	0.141	0.097	2,069,949
10	EPCSx10	50	7,172	0.520	0.249	0.177	32,912,259
11	dinox10H	100	6,000	0.299	0.155	0.125	90,680,222
12	DSAS	200	10,260	0.361	0.185	0.122	7,454,843
13	2ms7dx10_H	100	14,000	0.613	0.401	0.263	2,461,277
14	DSASx10	50	25,650	0.970	0.663	0.494	101,255,334
15	EPCSx50	50	35,862	2.760	1.354	0.940	164,561,295

En la gráfica de la Figura 4.6, se observa una comparativa de los tiempos de procesamiento utilizando *HOK-Means* ejecutándose en uno, dos y tres nodos. En la solución de cada *dataset*, los tiempos de procesamiento se reducen al incrementar el número de nodos. Los marcados de color azul, son los ejecutados utilizando tres nodos y en cada caso es el menor.



ndk	600	717	1,200	1,400	1,434	2,565	2,800	2,869	5,130	6,000	7,172	10,260	14,000	25,650	35,862
k	100	50	200	100	100	50	200	200	100	100	50	200	100	50	50
d	5	7	5	7	7	45	7	7	45	50	7	45	70	45	7
n	1	2	1	2	2.04	1.14	2	2	1	1.2	20	1.14	2	11.4	102.46

Figura 4.6 Tiempo de solución al resolver *datasets* grandes con *HOK-Means* utilizando uno, dos y tres nodos

En la primera columna de la Tabla 4.12, se muestra el identificador de la prueba. En la segunda, el nombre del *dataset*. En la tercera, el número de grupos que se formaron. En la cuarta, el porcentaje de reducción de tiempo usando dos nodos con respecto a utilizar un sólo nodo. En la quinta, el porcentaje de reducción de tiempo usando tres nodos con respecto a utilizar dos nodos. En la sexta, el porcentaje de reducción de tiempo usando tres nodos con respecto a utilizar un sólo nodo. Todos los *datasets* redujeron su tiempo de procesamiento cada vez que se incrementaron los nodos.

Tabla 4.12 Porcentajes de reducción de tiempo al resolver los *datasets* con uno, dos y tres nodos

Id.	***dataset***	***k***	***%*** t_{p_2n} ***vs*** t_{p_1n}	***%*** t_{p_3n} ***vs*** t_{p_2n}	***%*** t_{p_3n} ***vs*** t_{p_1n}
1	Dino	100	57.7	26.8	69.0
2	EPCS	50	48.5	30.5	64.2
3	Dino	200	**60.9**	30.9	**73.0**
4	2ms7d	100	54.3	31.6	68.7
5	EPCS	100	46.5	32.8	64.0
6	DSAS	50	49.1	33.7	66.3
7	DSAS	100	48.7	33.4	65.8
8	2ms7d	200	46.3	32.4	63.7
9	EPCS	200	47.8	30.8	63.9
10	EPCSx10	50	52.1	29.1	66.1
11	dinox10H	100	48.1	19.6	58.3
12	DSAS	200	48.8	33.9	66.2
13	2ms7dx10_H	100	34.6	**34.5**	57.1
14	DSASx10	50	31.6	25.5	49.0
15	EPCSx50	50	51.0	30.5	65.9

Al resolver *datasets* grandes los algoritmos secuenciales tienen la limitante de procesar sólo la cantidad de objetos que se pueden cargar en la memoria, antes de generar un error por desbordamiento de memoria. *HOK-Means* utiliza el parámetro *chunksize* del método *read_csv* de la librería *Pandas* de *Python*, la cual permite procesar *datasets* mayores a la capacidad de memoria principal de los equipos (Ver Anexo C).

En la Tabla 4.13, se muestran los *datasets* de la Tabla 4.12, que se lograron ejecutar con los algoritmos *K-Means* estándar y *O-K-Means* secuencial, sin que se generara un desbordamiento de memoria. En la primera columna, se presenta el identificador que lo relaciona con la Tabla 4.11. En la segunda y tercera, el tiempo en horas de procesamiento de los algoritmos *K-Means* estándar y *O-K-Means,* respectivamente. En la cuarta y quinta, el porcentaje de reducción de tiempo de *HOK-Means* usando tres nodos con respecto a *K-Means* estándar y *O-K-Means,* respectivamente. En la sexta, séptima y octava se presentan los valores del *Sp* de *HOK-Means* con uno, dos y tres nodos, con respecto al *O-K-Means*. En la novena, décima y onceava el valor de la Eficiencia paralela con uno, dos y tres nodos, con respecto al *O-K-Means*.

En la Figura 4.7, se observa una comparación entre el tiempo de procesamiento para solucionar los *datasets* procesados de manera secuencial (*O-K-Means*) y paralela (*HOK-Means*) empleando uno, dos y tres nodos. En todos los casos, el menor tiempo de procesamiento lo obtuvo la propuesta desarrollada con esta investigación, con tres nodos. En la Figura 4.8, se muestra una comparativa entre los *speedups* obtenidos con más de un nodo.

Tabla 4.13 Resultados % de reducción de tiempo, *Sp* y *Ep*, con respecto a *K-Means* y *O-K-Means*

Id.	t	t_o	% t_{p_3n} vs t_{sec}	% t_{p_3n} vs t_{std}	*Sp* 1Nodo	*Sp* 2Nodos	*Sp* 3Nodos	*Ep* 1Nodo	*Ep* 2Nodos	*Ep* 3Nodos
1	5.656	0.250	90.36	99.57	3.21	7.59	10.37	0.40	0.47	0.43
2	1.468	0.209	91.02	98.72	3.98	7.74	11.13	0.50	0.48	0.46
3	17.428	0.520	91.87	**99.76**	3.32	8.49	12.29	0.42	0.53	0.51
4	15.115	0.616	93.13	99.72	4.55	9.95	14.55	0.57	0.62	0.61
5	4.949	0.551	92.64	99.18	4.89	9.13	13.58	0.61	0.57	0.57
6	5.844	0.622	94.53	99.42	6.17	12.12	18.30	0.77	0.76	0.76
7	12.346	1.414	95.48	99.48	**7.56**	**14.74**	**22.14**	**0.95**	**0.92**	**0.92**

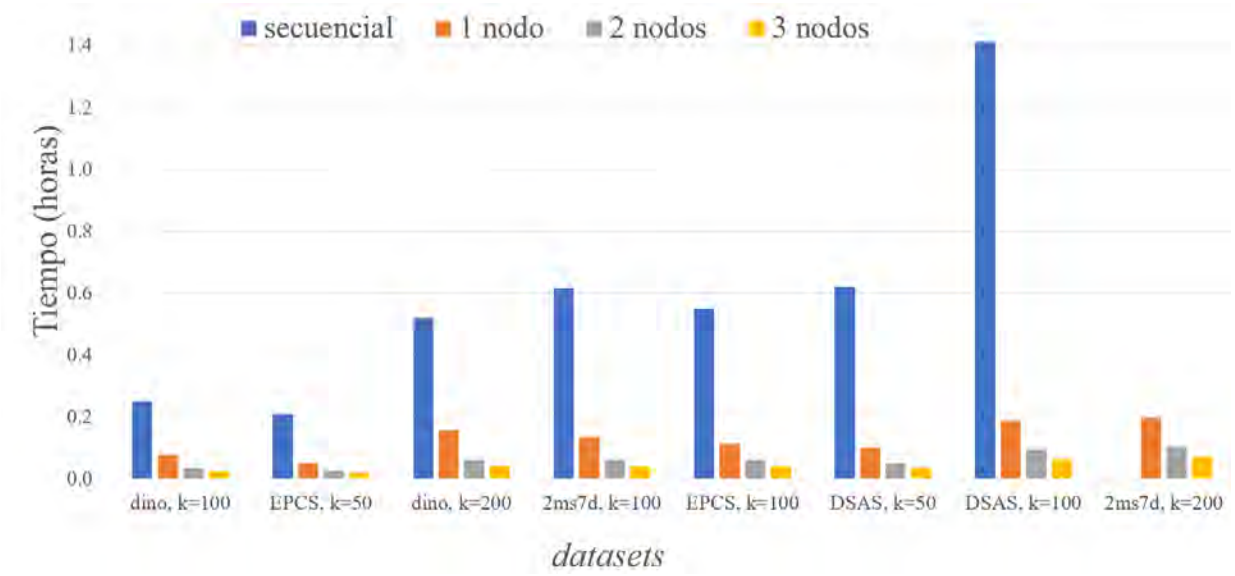


Figura 4.7 Tiempo de solución de *datasets* grandes con *O-K-Means* secuencial y *HOK-Means* utilizando uno, dos y tres nodos

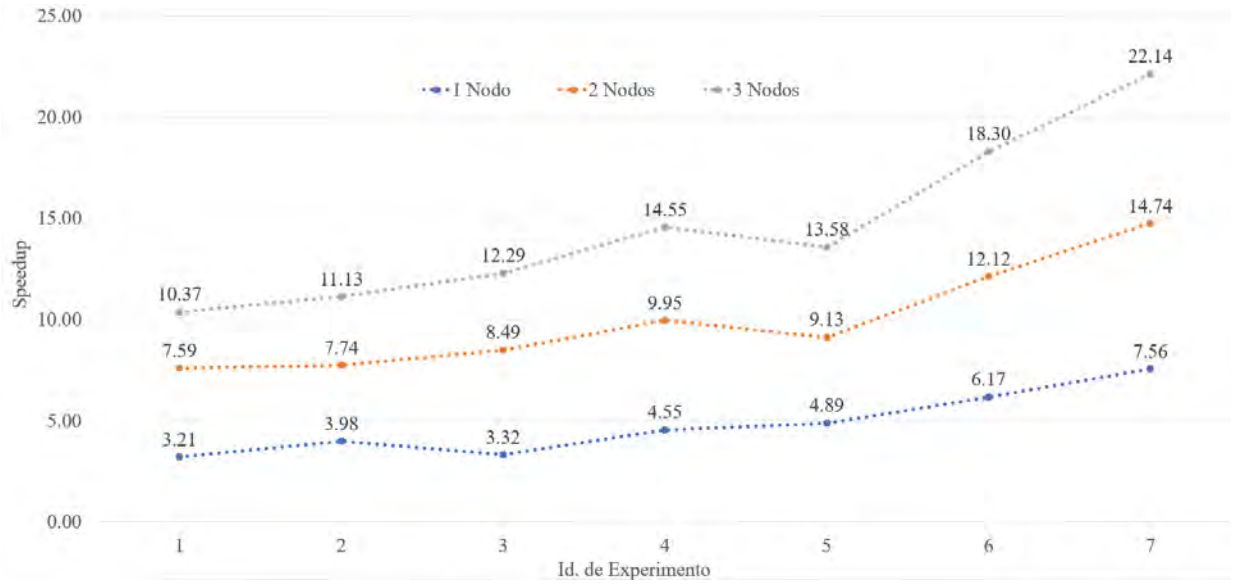


Figura 4.8 *Speedup* obtenido al resolver *datasets* grandes con *HOK-Means* usando uno, dos y tres nodos

4.7 Análisis de resultados y observaciones

En esta investigación, se propuso desarrollar un algoritmo llamado *HOK-Means*. El objetivo de esta heurística, es reducir el tiempo de solución, cuando se utilizan *datasets* grandes como los que se presenta en *Big Data*.

La heurística *HOK-Means* consiste en paralelizar *O-K-Means*, la cual es una variante del algoritmo *K-Means*. *HOK-Means* ejecuta de manera paralela y eficiente las fases de convergencia y clasificación de *O-K-Means*, conservando su calidad de agrupamiento. La disminución de tiempo se observa incluso usando un solo equipo.

Los mejores resultados se obtuvieron con los *datasets* más grandes, es decir, con los *datasets* donde el indicador *ndk* es alto y en los cuales se tenían mayor número de dimensiones. Como se muestra en la Tabla 4.13, donde el promedio de Eficiencia paralela para los *datasets* más grandes tiene un promedio de 0.61.

Los resultados obtenidos con la solución de los experimentos diseñados para validar la heurística *HOK-Means*, son muy alentadores y respaldan significativamente el uso de diferentes técnicas de paralelismo en heurísticas previamente mejoradas.

Con base en el análisis de los resultados experimentales, se tienen las siguientes observaciones:

a) La calidad de agrupamiento y las salidas del algoritmo secuencial, son iguales que las salidas del algoritmo con un enfoque paralelo/distribuido en un menor tiempo, incluso usando el mismo equipo.

b) En todos los experimentos realizados, la reducción promedio de calidad del agrupamiento fue menor al 1%. Los incisos a y b muestran que la elección de *O-K-Means* como algoritmo base fue adecuada.

c) Con *datasets* grandes y utilizando un enfoque paralelo y distribuido, los tiempos de procesamiento se reducen más si se incrementan el número de nodos.

d) Dados dos *datasets* con el mismo valor del indicador *ndk*, el *dataset* con mayor número de atributos tendrá un *speedup* más alto.

e) Para un mismo equipo con varios procesadores se muestra que con el enfoque paralelo y distribuido se pueden resolver *datasets* mayores que con el enfoque secuencial tradicional.

f) El *dataset* más grande que se resolvió con *HOK-Means* fue *DSASx50* y requirió de 7.25 horas utilizando tres nodos. Dicho *dataset* tiene los siguientes valores de n=57,000,000; d=45; k=200. El archivo *DSASx50* tiene un tamaño de 19.7GB.

4.8 Discusión

Los resultados de los tres experimentos del Capítulo 4, muestran que el algoritmo *HOK-Means* es más rápido que O-*K-Means* al resolver grandes instancias. La discusión correspondiente de *O-K-Means* respecto a otras investigaciones se justifica en la investigación de la Dra. Almanza en [6, 20].

Por otra parte, una forma de comparar *HOK-Means,* con otros trabajos similares, es mediante la Eficiencia paralela. Sin embargo, la falta de información para obtener su valor de Eficiencia paralela, dificulta una buena comparación, ver Tabla 2.1. Son pocas las investigaciones similares que involucran algoritmos paralelos de propósito general, basados en una variante o mejora secuencial. Las tres investigaciones más cercanas son descritas en la Sección 2.2.2.

En la literatura el valor de Eficiencia paralela mas alta es de 0.97, alcanzado por un algoritmo paralelo, de propósito general, basado en *K-Means.* el cual fue ejecutado en una supercomputadora. *HOK-Means* muestra una Eficiencia de 0.94 con respecto a *O-K-Means*, usando 3 equipos convencionales. A su vez *O-K-Means*, tiene una reducción de tiempo de hasta 93%, con respecto al *K-Means* estándar.

Dos trabajos que están basados en variantes y son de propósito general y además usan equipo convencional como *HOK-Means* reportan una Eficiencia paralela de 0.64 y 0.5, por lo cual *HOK-Means* es superior.

Capítulo 5

Conclusiones y trabajos futuros

No hay nada que temer en la vida, únicamente se debe entender.
Ahora es tiempo de entender más, para temer menos.
Marie Curie

Como resultado de la presente investigación se mostró que es factible reducir el tiempo de procesamiento del algoritmo *K-Means* para la solución de grandes *datasets*, mediante un nuevo enfoque basado en la selección de una variante eficiente del algoritmo *K-Means*. Específicamente, a dicha variante implementada bajo el paradigma de programación paralela y distribuida y se le denominó *Hybrid O-K-Means* (*HOK-Means*). Con base en los resultados obtenidos, se observó que el algoritmo *HOK-Means* redujo el tiempo de solución del algoritmo *O-K-Means*, cuando se resolvieron *datasets* de gran tamaño.

En este capítulo, se recogen las conclusiones derivadas de la investigación y algunas recomendaciones para continuar en futuros trabajos.

5.1 Conclusiones

La variante *HOK-Means*, implementada en esta investigación, reduce el tiempo de procesamiento al solucionar *datasets* grandes del tipo *Big Data*, conservando la calidad de agrupamiento del algoritmo *O-K-Means*, el cual muestra una reducción poco significativa en la calidad de solución.

Para validar los resultados del enfoque de solución, se diseñaron un conjunto de experimentos, resolviendo *datasets* sintéticos y reales, algunos de ellos de gran tamaño, usando los recursos de una red de computadoras. Para valorar los resultados se utilizaron como indicadores el valor de la función objetivo, el tiempo de procesamiento, el *speedup* y la eficiencia paralela.

Las principales contribuciones de esta investigación son las siguientes:

1. Una adecuada selección de la variante O-*K-Means*, la cual mostró ser factible de paralelizar, con un alto índice de eficiencia paralela.
2. Un favorable diseño paralelo. El cual evita sobrecarga en la memoria, tiene un bajo costo de envío de datos a través de la red y una mínima escritura de datos en disco duro.
3. Una fácil implementación, la cual es una característica del lenguaje de programación Python. Sin embargo, su diseño puede fácilmente implementarse en otros lenguajes o arquitecturas de programación paralela.

4. Un diseño de experimentos, que permite validar correctamente la eficiencia de una nueva variante. Además, se usaron métricas como *speedup* y eficiencia paralela que permiten realizar el *benchmark* con otros trabajos similares publicados,
5. La variante *HOK-Means* es escalable, se puede configurar para aprovechar todos los recursos de *hardware* disponibles en una red de computadoras, por ejemplo: los procesadores, la memoria principal, el disco duro de cada computadora disponible para la solución del *dataset*. Se puede utilizar tanto en equipos convencionales como en equipos más potentes, el único requerimiento es que se ejecuten las librerías de *Python*.
6. Los mejores resultados en eficiencia paralela y distribuida se obtuvieron con *datasets* con un indicador *ndk* grande.
7. Con base en los resultados experimentales, se observó que *HOK-Means* redujo el tiempo de solución en todos los *datasets* cuyo índice *ndk* fue superior a 16 millones. Usando una sola computadora, el *speedup* en los *datasets* sintéticos fue desde 1.6 hasta 6.1, con una eficiencia paralela de 0.2 hasta 0.76. En la solución de *datasets* reales, el *speedup* fue de 1.7 hasta 7.54, con una eficiencia paralela de hasta 0.94. Usando tres computadoras, se pudo obtener un *speedup* de hasta 22.2 con una eficiencia paralela de 0.92. Usando tres computadoras, *HOK-Means* redujo en promedio el 64.1% de tiempo de solución, que usando solo una. Comparado con otras investigaciones similares *HOK-Means* mostró superior eficiencia paralela que otras variantes paralelas de propósito general.
8. Con relación a la calidad de solución en promedio, en los *datasets* sintéticos hubo una reducción de la calidad de -0.41% y en los reales de -0.27%. Para los cuatro *datasets* más grandes la pérdida de calidad fue de 0.15%. Como se puede observar la reducción de la calidad de la solución es poco significativa.
9. Usando el mismo recurso, una sola computadora, *HOK-Means* resolvió *datasets* que el algoritmo *O-K-Means* no resolvió.

5.2 Trabajos futuros

Con el fin de dar continuidad a esta investigación, se sugiere que, en investigaciones posteriores, se desarrollen los siguientes temas:

a) Adecuar el algoritmo *HOK-Means* para usarlo en la nube, y compararlo con los algoritmos usados actualmente.
b) Combinar *O-K-Means* con algoritmos que prometan eficiencia en la calidad del agrupamiento, por ejemplo, *K-Means++* y paralelizarlo y distribuirlo.

Referencias

[1] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognition Letters,* vol. 31, no. 8, pp. 651-666, Jun. 2010, doi: doi.org/10.1016/j.patrec.2009.09.011.

[2] T. Fawcett and F. Provost, "Introduction: Data-Analytic Thinking," in *Data Science for Business [What you need to know about data mining and data-analytic thinking]*, T. Fawcett Ed., 1st ed. Sebastopol, CA, USA: O'Reilly, 2013, ch. 1, pp. 1-16.

[3] J. W. Foreman, *Data smart: using data science to transform information into insight*, 1st ed. (Wrox professional guides). Hoboken, New Jersey, USA: John Wiley & Sons, 2014, pp. xiii-xx.

[4] K. Kambatla, G. Kollias, V. Kumar, and A. Grama, "Trends in big data analytics," *Journal of Parallel and Distributed Computing,* vol. 74, no. 7, pp. 2561-2573, Jul. 2014, doi: doi.org/10.1016/j.jpdc.2014.01.003.

[5] Y. P. Raykov, A. Boukouvalas, F. Baig, and M. A. Little, "What to Do When K-Means Clustering Fails: A Simple yet Principled Alternative Algorithm," *PLOS ONE,* vol. 11, no. 9, pp. 1-28, Sep. 2016, doi: 10.1371/journal.pone.0162259.

[6] J. Pérez-Ortega, N. N. Almanza-Ortega, and D. Romero, "Balancing effort and benefit of K-means clustering algorithms in Big Data realms," *PLOS ONE,* vol. 13, no. 9, pp. 1-19, Sep. 2018, doi: 10.1371/journal.pone.0201874.

[7] C. O'Neil and R. Schutt, C. N. Mike Loukides, Ed. *Doing data science: Straight talk from the frontline*, 1st. ed. Sebastopol, CA, USA: O'Reilly, 2013.

[8] D. Judd, P. K. McKinley, and A. K. Jain, "Large-scale parallel data clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 20, no. 8, pp. 871-876, Aug. 1998, doi: 10.1109/34.709614.

[9] K. Stoffel and A. Belkoniene, "Parallel k/h-Means Clustering for Large Data Sets," in *European Conf. on Parallel Processing*, Berlin, Heidelberg, 1999: Springer, in Euro-Par'99 Parallel Processing, pp. 1451-1454.

[10] G. R. Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming*, 1st. ed. USA: Addison Wesley Longman, Inc., 2000, pp. 1-31.

[11] A. Grama, G. Karypis, V. Kumar, and A. Gupta, "Principles of Parallel Algorithm Design," in *Introduction to parallel computing*, 2nd ed.: Addison-Wesley, 2003, ch. 3, pp. 92-151.

[12] G. T. Castro, L. E. Zárate, C. N. Nobre, and H. C. Freitas, "A Fast Parallel K-Modes Algorithm for Clustering Nucleotide Sequences to Predict Translation Initiation Sites," *Journal of Computational Biology,* vol. 26, no. 5, pp. 442-456, 2019.

[13] M. Al-Ayyoub, M. Al-andoli, Y. Jararweh, M. Smadi, and B. Gupta, "Improving fuzzy C-mean-based community detection in social networks using dynamic parallelism," *Computers & electrical engineering,* vol. 74, pp. 533-546, Jan. 2018, doi: doi.org/10.1016/j.compeleceng.2018.01.003.

[14] S. Pan, J. Qiao, and L. Zhu, "Application of Parallel Clustering Algorithm Based on R in Power Customer Classification," in *4th Int. Conf. on Cloud Computing and Big Data Analysis*, Chengdu, China, Apr. 12-15, 2019: IEEE pp. 165-169.

[15] J. Jiménez, R. O. Klenzi, and A. Malberti, "Análisis de desempeño de una Implementación del algoritmo K-means en CUDA y OMP," in *XVII Workshop de Investigadores en Ciencias de la Computación*, Salta, Argentina, Apr. 16-17, 2015.

[16] N. Francis and J. Mathew, "Implementation of parallel clustering algorithms using Join and Fork model," in *Online Int. Conf. on Green Engineering and Technologies*, 2016: IEEE, pp. 1-5.

[17] Z. Ansari, A. Afzal, and T. H. Sardar, "Data categorization using hadoop MapReduce-based parallel K-means clustering," *Journal of The Institution of Engineers (India): Series B,* vol. 100, no. 2, pp. 95-103, 2019.

[18] T. H. Sardar and Z. Ansari, "An analysis of distributed document clustering using MapReduce based K-means algorithm," *Journal of The Institution of Engineers (India): Series B,* vol. 101, no. 6, pp. 641-650, 2020.

[19] A. E. Top, F. Ş. Torun, and H. Kaya, "Parallel and distributed image segmentation based on colors using K-means clustering algorithm," in *5th Int. Conf. on Engineering Sciences*, Sep. 2019, Ankara, Turkey, pp. 302-304.

[20] N. N. Almanza Ortega, "Desarrollo de heurísticas para la mejora del algoritmo K-Means en las fases de clasificación y convergencia," Tesis de doctorado, Dpto. Ciencias Computacionales, CENIDET, Cuernavaca, Mor., 2018.

[21] D. Dua and C. Graff. "UCI Machine Learning Repository." http://archive.ics.uci.edu/ml (accessed 20 January 2022, 2022).

[22] J. Pérez-Ortega, N. N. Almanza-Ortega, A. Vega-Villalobos, R. Pazos-Rangel, J. C. Zavala-Diaz, and A. Martínez-Rebollar, "The K-Means Algorithm Evolution," in *Introduction to Data Science and Machine Learning*, 2019.

[23] C. F. M. Calderón, "Caracterización de dos mejoras del estado del arte del algoritmo K-Means orientadas a la solución de grandes instancias," Tesis de maestría, Dpto. Ciencias Computacionales, CENIDET, Cuernavaca, Mor., 2022.

[24] A. M. Fahim, A. M. Salem, F. A. Torkey, and M. A. Ramadan, "An efficient enhanced k-means clustering algorithm," *Journal of Zhejiang University-SCIENCE A,* vol. 7, no. 10, pp. 1626-1633, Oct. 2006, doi: 10.1631/jzus.2006.A1626.

[25] C. A. Navarro, N. Hitschfeld-Kahler, and L. Mateu, "A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures," *Communications in Computational Physics,* vol. 15, no. 2, pp. 285-329, 2014, doi: 10.4208/cicp.110113.010813a.

[26] X. Li and Z. Fang, "Parallel clustering algorithms," *Parallel Computing,* vol. 11, no. 3, pp. 275-290, Aug. 1989, doi: doi.org/10.1016/0167-8191(89)90036-7.

[27] I. S. Dhillon and D. S. Modha, "A data-clustering algorithm on distributed memory multiprocessors," in *Large-scale parallel data mining*: Springer, 2002, pp. 245-260.

[28] Z. Wang, A. Xu, Z. Zhang, C. Wang, A. Liu, and X. Hu, "The parallelization and optimization of K-means algorithm based on spark," in *15th Int. Conf. on Computer Science & Education* 2020: IEEE, pp. 457-462.

[29] R. Azimi, H. Sajedi, and M. Ghayekhloo, "A distributed data clustering algorithm in P2P networks," *Applied Soft Computing,* vol. 51, pp. 147-167, Feb. 2017, doi: doi.org/10.1016/j.asoc.2016.11.045.

[30] M. Al-Ayyoub, Q. Yaseen, M. A. Shehab, Y. Jararweh, F. Albalas, and E. Benkhelifa, "Exploiting gpus to accelerate clustering algorithms," in *13th Int. Conf. of Computer Systems and Applications*, 2016: IEEE, pp. 1-6.

[31] C. Lutz, S. Breß, T. Rabl, S. Zeuch, and V. Markl, "Efficient k-means on GPUs," in *14th Int.Workshop on Data Management on New Hardware*, Jun. 2018, pp. 1-3.

[32] A. Hadian and S. Shahrivari, "High performance parallel k-means clustering for disk-resident datasets on multi-core CPUs," *The Journal of Supercomputing,* vol. 69, no. 2, pp. 845-863, 2014.

[33] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "FPGA implementation of K-means algorithm for bioinformatics application: An accelerated approach to clustering Microarray data," in *Conf. on Adaptive Hardware and Systems (AHS)*, Jun. 2011, pp. 248-255, doi: 10.1109/AHS.2011.5963944.

[34] F. Jia, C. Wang, X. Li, and X. Zhou, "SAKMA: Specialized FPGA-Based Accelerator Architecture for Data-Intensive K-Means Algorithms," in *Int. Conf. on Algorithms and Architectures for Parallel Processing*, 2015, vol. 9529: Springer, Cham., pp. 106-119, doi: DOI: 10.1007/978-3-319-27122-4_8.

[35] A. Fahad *et al.*, "A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis," *IEEE Transactions on Emerging Topics in Computing,* vol. 2, no. 3, pp. 267-279, 2014, doi: 10.1109/TETC.2014.2330519.

[36] L. Wan, G. Zhang, H. Li, and C. Li, "A novel bearing fault diagnosis method using spark-based parallel ACO-K-Means clustering algorithm," *IEEE Access,* vol. 9, pp. 28753-28768, 2021.

[37] D. Xia, B. Wang, Y. Li, Z. Rong, and Z. Zhang, "An efficient MapReduce-based parallel clustering algorithm for distributed traffic subarea division," *Discrete Dynamics in Nature and Society,* vol. 2015, 2015.

[38] P. Mackey and R. R. Lewis, "Parallel k-means++ for multiple shared-memory architectures," in *45th Int. Conf. on Parallel Processing (ICPP)*, 2016: IEEE, pp. 93-102, doi: DOI 10.1109/ICPP.2016.18.

[39] T. Yu *et al.*, "Large-Scale Automatic K-Means Clustering for Heterogeneous Many-Core Supercomputer," *IEEE Transactions on Parallel and Distributed Systems,* vol. 31, no. 5, pp. 997-1008, 2019.

[40] A. K. Tripathi, K. Sharma, M. Bala, A. Kumar, V. G. Menon, and A. K. Bashir, "A parallel military-dog-based algorithm for clustering big data in cognitive industrial internet of things," *IEEE Transactions on Industrial Informatics,* vol. 17, no. 3, pp. 2134-2142, 2020.

[41] S. S. Bandyopadhyay, A. K. Halder, P. Chatterjee, M. Nasipuri, and S. Basu, "Hdk-means: Hadoop based parallel k-means clustering for big data," in *Calcutta Conf.(CALCON)*, 2017: IEEE, pp. 452-456.

[42] X.-Y. Peng, Y.-B. Yang, C.-D. Wang, D. Huang, and J.-H. Lai, "An efficient parallel nonlinear clustering algorithm using mapreduce," in *Int. Parallel and Distributed Processing Symp. Workshops (IPDPSW)*, 2016: IEEE, pp. 1473-1476, doi: DOI 10.1109/IPDPSW.2016.7.

[43] S. Ling and Q. Yunfeng, "Optimization of the distributed K-means clustering algorithm based on set pair analysis," in *8th Int.Congr. on Image and Signal Processing (CISP)*, 2015: IEEE, pp. 1593-1598.

[44] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on mapreduce," in *IEEE Int. Conf. on cloud computing*, 2009: Springer, pp. 674-679.

[45] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "NP-hardness of Euclidean sum-of-squares clustering," *Machine learning,* vol. 75, no. 2, pp. 245-248, 2009. [Online]. Available: doi.org/10.1016/0167-8191(89)90036-7.

[46] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, "The planar k-means problem is NP-hard," *Theoretical Computer Science,* vol. 442, pp. 13-21, 2012.

[47] A. Kaur and A. Datta, "A novel algorithm for fast and scalable subspace clustering of high-dimensional data," *Journal of Big Data,* vol. 2, no. 1, p. 17, Aug. 2015, doi: 10.1186/s40537-015-0027-y.

[48] L. Kleinrock and J.-H. Huang, "On parallel processing systems: Amdahl's law generalized and some results on optimal design," *Performance Modeling for Computer Architects,* vol. 11, p. 66, 1995.

[49] "Parallel Python Documentation." https://www.parallelpython.com/ (accessed 20 January 2022, 2022).

[50] "Pandas." https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html (accessed 10/01/2022, 2022).

Anexo A. Algoritmo *O-K-Means*

En el Algoritmo 2, se describe el algoritmo *O-K-Means*, el cual, a diferencia del *K-Means* estándar, recibe como entrada adicional el valor determinado del umbral. En la etapa de Clasificación se calcula $\gamma_r = 100(v_r/n)$, donde *r* es la iteración; v_r es el número de puntos que cambiaron de grupo con respecto a la iteración anterior. En la etapa de convergencia, en lugar de verificar si el conjunto de centroides *M* no cambió, se compara si γ_r es menor al umbral determinado. Para más información ver la tesis[20] o el artículo[6].

Algoritmo 2: *O-K-Means*	
1	Inicialización:
2	$N := \{x_1, \ldots x_n\}$;
3	$M := \{\mu_1, \ldots \mu_k\}$;
4	$U :=$ Valor del umbral;
5	Clasificación:
6	Para $x_i \in N$ y $\mu_k \in M$ {
7	Calcular la distancia Euclidiana de cada x_i a los *k* centroides;
8	Asignar el objeto x_i al centroide μ_k más cercano;
9	Calcular γ; }
10	Cálculo de centroides:
11	Calcular el centroide μ_k;
12	Convergencia:
13	Sí ($\gamma \leq U$):
14	Detener el algoritmo;
15	En caso contrario:
16	Ir a Clasificación
17	Fin del algoritmo

Anexo B. Descripción general de Parallel Python

Parallel Python es un módulo del interprete *Python* que proporciona un mecanismo para la ejecución paralela de código de Python en **SMP** (sistemas con múltiples procesadores o núcleos) y **clústeres** (computadoras conectadas a través de la red). Es ligero, fácil de instalar e integrar con otro software de *Python*. ***Parallel Python*** es un módulo de código abierto y multiplataforma escrito en Python puro. Para más información véase [49].

Características:

a) Ejecución paralela de código *Python* en SMP y clústeres,
b) Técnica de paralelización basada en trabajos fácil de entender e implementar (fácil de convertir aplicaciones en serie en paralelo),
c) Detección automática de la configuración óptima (de forma predeterminada, la cantidad de procesos de trabajo se establece en la cantidad de procesadores efectivos),
d) Asignación dinámica de procesadores (la cantidad de procesos de trabajo se puede cambiar en tiempo de ejecución),
e) Baja sobrecarga para trabajos posteriores con la misma función (se implementa el almacenamiento en caché transparente para disminuir la sobrecarga),
f) Equilibrio de carga dinámico (los trabajos se distribuyen entre procesadores en tiempo de ejecución),
g) Tolerancia a fallas (si uno de los nodos falla, las tareas se reprograman en otros),
h) Auto-descubrimiento de recursos computacionales,
i) Asignación dinámica de recursos computacionales (consecuencia del descubrimiento automático y tolerancia a fallas),
j) Autenticación basada en SHA para conexiones de red,
k) Portabilidad e interoperabilidad multiplataforma (Windows, Linux, Unix, Mac OS X),
l) Portabilidad e interoperabilidad entre arquitecturas (x86, x86-64, etc.),
m) Fuente abierta.

Anexo C. Parámetro *chunksize* de método *read_csv*()

Pandas en una herramienta de análisis de datos de código abierto flexible y fácil de usar construida sobre *Python*, lo que facilita la importación y visualización de datos de diferentes formatos como .csv, .tsv , .txt. En esta tesis se generaron y leyeron archivos .csv y .txt. El método utilizado para leer archivos .CSV es read_csv().

El método *read_csv*() tiene muchos parámetros, pero el que interesa es *chunksize*. Técnicamente, es el número de filas leídas a la vez en un archivo por pandas se conoce como tamaño de fragmento. Supongamos que, si el tamaño de fragmento es 100, los pandas cargarán las primeras 100 filas. El objeto devuelto no es un marco de datos sino un *TextFileReader* que debe iterarse para obtener los datos.

chunksize: int, opcional Retorna el objeto TextFileReader para la iteración, ver Figura 8.1. Consulte los documentos de *IO Tools* [50] para obtener más información sobre *iterator* y *chunksize*. A forma de ejemplo se muestra la porción de código *Python*, donde se utilizó el parámetro *chunksize*:

```
if cpuActual  != nTotalCPU:
      try:
         src_c = pandas.read_csv(ruta,header=None, skiprows=lineaInicio-1,
                                 nrows = bloque, chunksize=59999, low_memory=False)

      except Exception as e:
            print("fallo al leer ......1: "+bd+ "\n N_CPU: "+str(cpuActual))
            print("\nruta"+ruta +"\n linea Inicio: "+str(lineaInicio-1)+"\nBloque: "+str(bloque))
            print(e)
```

Figura 0.1 Ejemplo de uso del parámetro *chunksize* en código de lenguaje *Python*

```
for src in src_c:
    dic_r, mem,sZ=file_result(src,centroides,columnas,grupos)

    dic_res=pandas.concat([dic_res,dic_r],axis=0)
    mem_res=pandas.concat([mem_res,mem],axis=0)
    sZ_res=sZ_res+sZ
```

Figura 0.2 Iteración para el objeto *TextFileReader* del lenguaje *Python*