DMTree: Un Nuevo Método de Indexación para Hallar Similitudes en Grandes Conjuntos de Vectores

Phuc Do[1]
Faculty of Information Technology
University of Information Technology (UIT)
VNU-HCM
Ho Chi Minh City
Vietnam

Trung Phan Hong[2]
Faculty of Science and Information Technology, University of Information Technology (UIT), VNU-HCM, Faculty of Information Technology, Hoa Sen University
Ho Chi Minh City, Vietnam

Huong Duong To[3]
Faculty of Information Technology
Hoa Sen University
Ho Chi Minh City
Vietnam

***Abstract*— En un conjunto de vectores, para encontrar similitudes calcularemos las distancias desde el vector consultado a todos los demás vectores. En un conjunto de vectores de gran tamaño, calcular demasiadas distancias como en el caso anterior lleva mucho tiempo. Así que tenemos que encontrar una manera de calcular menos distancias y la estructura MTree es la técnica que necesitamos. La estructura MTree es una técnica de indexación de conjuntos de vectores basada en una distancia definida. Podemos resolver eficazmente los problemas de búsqueda de similitudes utilizando la estructura MTree. Sin embargo, la estructura MTree se construye en un solo ordenador, por lo que la capacidad de indexación es limitada. Hoy en día, los conjuntos de vectores de gran tamaño, que no caben en un ordenador, son cada vez más numerosos. La estructura MTree no es capaz de indexar estos grandes conjuntos de vectores. Por lo tanto, en este trabajo, presentamos un nuevo método de indexación, ampliado a partir de la estructura MTree, que puede indexar grandes conjuntos de vectores. Además, también realizamos experimentos para probar el rendimiento de este nuevo método.**

Palabras Clave—MTree; DMTree; spark; distributed k-NN query;distributed range query

I. INTRODUCCION

En el mundo real, las aplicaciones de grafos aparecen por todas partes y los grafos son cada vez más grandes. Los grafos con millones o miles de millones de vértices son muy populares. Sin embargo, la aplicación de cálculos matemáticos y algoritmos de aprendizaje automático en grafos es limitada y muy difícil. Por lo tanto, una nueva y prometedora técnica de procesamiento de grafos, que despierta un gran interés en los círculos de investigación, es la técnica de incrustación de grafos [1]-[3]. La tecnología de incrustación de grafos se basa en la tecnología word2vec [4][5]. Word2vec es una tecnología que convierte palabras en vectores. Esta tecnología ha ayudado a resolver una serie de problemas de Procesamiento del Lenguaje Natural con mucha mayor precisión que antes. En la incrustación de grafos, cada vértice de un grafo se convierte en un vector de 64, 128, 256... dimensiones, cada una de las cuales es un número real. Para conservar toda la información posible del grafo original, el número de dimensiones del conjunto vectorial es tanto mayor. Hoy en día, los grafos grandes son muy populares, por lo que los conjuntos vectoriales grandes también lo son.

Por otro lado, para encontrar similitudes en conjuntos de vectores, debemos calcular distancias desde el vector consultado a todos los demás vectores. En un conjunto de vectores grande, no podemos calcular demasiadas distancias como en el caso anterior. Durante el proceso de investigación, nos dimos cuenta de que la estructura MTree es una técnica de indexación de conjuntos de vectores basada en una distancia definida. Usando la estructura MTree, podemos resolver los problemas de encontrar similitudes de forma efectiva [6]. Dado que la estructura MTree sólo se construye en un ordenador, sólo puede indexar conjuntos vectoriales pequeños, donde todos los vectores pueden almacenarse en un ordenador. Hoy en día, los conjuntos de vectores grandes, donde los vectores están distribuidos en un cluster de ordenadores, son cada vez más populares. La estructura MTree no es capaz de indexar estos grandes conjuntos de vectores. Por ello, en este trabajo construimos la estructura MTree distribuida (DMTree para abreviar) extendiendo la estructura MTree en Spark, un famoso framework para el procesamiento distribuido, para indexar grandes conjuntos de vectores. Además, también realizamos experimentos con ambas estructuras para demostrar que el rendimiento de la estructura DMTree es mejor que el de la estructura MTree.

Las principales contribuciones de nuestro trabajo son las siguientes:

- Proponer un método para construir la estructura DMTree para indexar grandes conjuntos de vectores.
- Uso de la estructura DMTree para encontrar similitudes en grandes conjuntos de vectores.
- Presentación de experimentos para demostrar que la estructura DMTree es mejor que la estructura MTree.

Hay 6 secciones en este artículo que incluyen:
I. Introducción, II. Trabajos Relacionados, III. Preliminares,
IV. Metodología, V. Experimentos, and VI. Conclusiones y Trabajos Futuros.

II. TRABAJOS RELACIONADOS

Inspirados en [6], hay muchos trabajos sobre La implementación, el desarrollo de la estructura MTree y otras estructuras de indexación.
El autor en [7] ha propuesto la estructura MVPTree (la estructura

de árbol de múltiples puntos de ventaja) para particionar conjuntos de vectores. Se han realizado experimentos para comparar la estructura MVPTree y la estructura MTree.

El autor en [8] ha construido un framework para encontrar similitudes, donde los datos son indexados localmente usando la estructura MTree. Este trabajo ha utilizado una arquitectura de super-pares, donde los super-pares son responsables del enrutamiento de las consultas, para soportar la escalabilidad y la eficiencia en la búsqueda de similitudes.

El autor de [9] ha investigado una estructura de árbol para indexar y consultar datos basados en una métrica. Este trabajo también ha realizado experimentos para comparar su método con otros, como la estructura MMTree y la estructura SliMTree. Este también ha propuesto una clasificación de los métodos de indexación.

Y más recientemente, [10] ha construido la estructura SuperMTree extendiendo la estructura MTree para indexar conjuntos vectoriales. Este trabajo ha propuesto un concepto generalizado de espacios métricos que son los espacios de subconjuntos métricos. Varias funciones de distancia métrica pueden extenderse a funciones de distancia de subconjuntos métricos.

La mayoría de los trabajos anteriores no se refieren a la indexación de grandes conjuntos vectoriales, donde los vectores están distribuidos en un cluster de ordenadores, excepto [8]. Sin embargo, [8] ha construido un nuevo framework para encontrar similitudes de forma distribuida. En este trabajo, no crearemos ningún framework, sólo utilizaremos Spark, que es el famoso framework para el procesamiento distribuido, para construir la estructura DMTree para indexar grandes conjuntos de vectores de forma efectiva.

II. PRELIMINARES

A. Definiciones de las consultas de similitud

En los conjuntos de vectores, la tarea común es encontrar similitudes. En concreto, solemos encontrar los k vectores más cercanos al vector consultado; o encontrar todos los vectores en el rango de radio r, y el centro es el vector consultado. Se trata de la consulta k-vecinos más cercanos (k-NN query para abreviar) y la consulta de rango [6][11]. A continuación, se definen:

1) Definición 1. K-nearest neighbors' query: Dado un conjuto de vectores S, una función distancia d, un vector consulta $v \in S$, y un numero entero $k \geq 1$. El k-nearest neighbors query $kNNQuery(v, k)$ selecciona k vectores en S que son cercanas a v.

2) Definición 2. Range query: Dado un conjunto de vecotores S , una función distancia d, un vector consulta $v \in S$, y un radio r. La range query $rangeQuery(v, r)$ selecciona todos los vectores v_i en S tales que $d(v_i, v) \leq r$.

B. La Estructura del MTree

La estructura MTree es una técnica de indexación de conjuntos de vectores basada en una función de distancia definida [6][13]. En términos de estructura interna, es un árbol equilibrado, pero no requiere reconstrucción periódica.

Un nodo de la estructura MTree puede contener como máximo C objetos, C se denomina la capacidad de los nodos. Los nodos hoja contienen objetos indexados. Los nodos restantes contienen objetos de enrutamiento.

El formato de un objeto de enrutamiento es el siguiente:

$$[\, O_r\,,\, r_r\,, d(O_r\,, O_r^p), ptr(T_r)\,]$$

Donde O_r representa el objeto de enrutamiento: $r_r \geq 0$ esta cubriendo el radio de O_r; $d(O_r, O_r^p)$ es la distancia entre O_r y O_r^p q uien es el padre de O_r: ptr (T_r) es la referencia al subárbol T_r que está cubriendo el árbol O_r.

El formato de un objeto indexado O_i es:

$$[O_i, d\ (O_i, O_i^p)]$$

Donde O_i rrepresenta el objeto indexado, $d(O_i, O_i^p)$ es la distancia entre O_i y O_i^p que es el padre del objeto O_i.

La Fig. 1 es un ejemplo de la estructura MTree con C = 3. Esta estructura MTree incluye:

- El nodo 1 es el nodo raíz que contiene dos objetos de enrutamiento O_1 y O_5.
- El nodo 2 y el nodo 3 son los nodos internos. Los nodos internos contienen objetos de enrutamiento. Por ejemplo, el nodo 2 contiene dos objetos de enrutamiento que son O_1 y O_8. En el nodo 2, considere el objeto de enrutamiento O_8, [O_8, 3.5, 1.3, $ptr(T_r)$];3.5 es el radio de cobertura de O_8; 1.3 es la distancia entre O_8 y O_1 (en el nodo 1) que es el objeto padre de O_8.
- Los nodos 4, 5, 6, 7, 8 son los nodos hoja. Los nodos hoja contienen objetos indexados. Por ejemplo, el nodo hoja 4 contiene tres objetos indexados que son: O_1, O_2 y O_4 . En el nodo 4, considere el objeto indexado O_2, [O_2, 2.5]; 2.5 es la distancia entre O_2 y O_1 (en el nodo 2) que es el objeto padre de O_2.

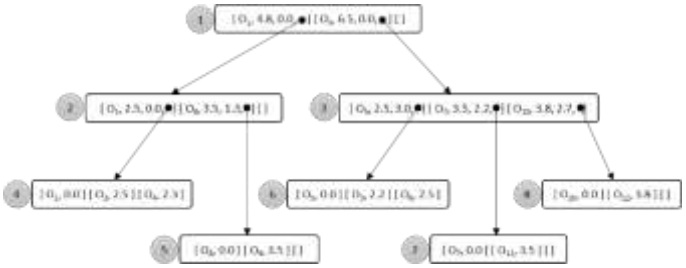


Fig. 1. Una instancia de la estructura MTree con C = 3

Consulte [6] [13] para obtener más información sobre la estructura MTree.

III. METODOLOGÍA

Al principio, construimos la estructura MTree ejecutándose en una computadora. Después de eso, extendemos la estructura MTree para construir la estructura DMTree que se ejecuta en un clúster Spark que consta de varias computadoras. También realizamos muchos experimentos en ambas estructuras para demostrar que la estructura DMTree es mejor que la estructura MTree. Los siguientes son detalles de nuestra solución.

A. Creación de un objeto DMTree

Al principio, creamos la clase MTree para representar la estructura MTree basada en [6][13]. La clase MTree tiene los siguientes métodos importantes:

- function insertObject(n: Node, v: Vector): Localiza el nodo hoja más adecuado en el subárbol del nodo N para almacenar un nuevo vector V. Es posible activar la división del nodo

hoja si el nodo hoja está lleno. Este método se utiliza para construir un objeto MTree a partir de un conjunto vectorial.

- function kNNQuery(v: Vector, k: Integer): Set[Vector]: ejecuta una consulta k-NN y devuelve k vectores que están más cerca de v.
- function rangeQuery(v: Vector, r: Double): Set[Vector]: Ejecuta una consulta de rango y devuelve todos los vectores tales que las distancias de v a ellos son menores o iguales a r.

Consulte [6] [13] para obtener más detalles sobre estos métodos.

A continuación, construimos la estructura DMTree definiendo la clase DMTree basada en la clase MTree. Un objeto DMTree incluye un conjunto de objetos MTree construidos a partir de un conjunto vectorial. El conjunto vectorial se distribuye en particiones de un RDD (Resilient Distributed Dataset), que es una estructura de datos fundamental de Apache Spark y es una colección tolerante a fallos de elementos que se pueden operar en paralelo [14] [15]. Las propiedades y métodos de la clase DMTree se muestran en la TABLA I. y la Tabla II.

El proceso de construcción de un objeto DMTree a partir de un conjunto vectorial se muestra en la Fig. 2. Primero, un conjunto vectorial se carga desde archivos distribuidos en un objeto RDD[Vector] en un clúster Spark [14]. En segundo lugar, se crea un objeto DMTree. A continuación, utilizando la transformación Map, que transforma un objeto RDD[X] en otro objeto RDD[Y] en paralelo (supongamos que X e Y son tipos dat a), asigna cada partición del objeto RDD[Vector] a un objeto MTree dentro del objeto DMTree. Se puede configurar el número de particiones de un objeto RDD[Vector], al igual que el número de objetos MTree de un objeto DMTree. El proceso de construcción de un objeto DMTree a partir de un conjunto vectorial se describe en el Algoritmo 1.

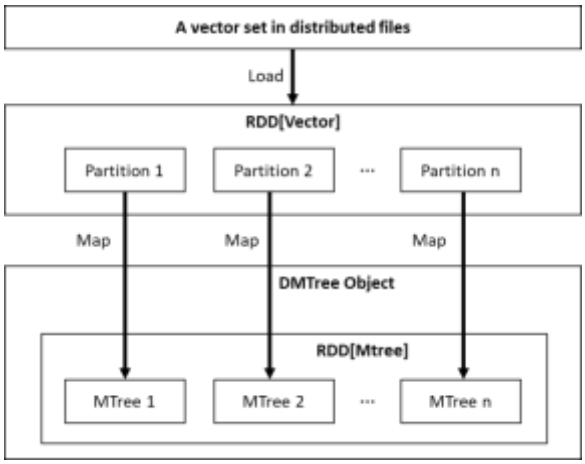
TABLA I. PROPERTIES DE LA DMTREE CLASS

Propiedades	Tipos de datos	Descripciones
C	Integer	La capacidad de los objetos DMTree. En esencia, esta propiedad es la capacidad de los nodos de Objetos MTree dentro de un objeto DMTree.
mtrees	RDD[MTree]	Contiene un conjunto de objetos MTree dentro de un objeto DMTree.

TABLA II. M ETHODS DE LA DMTREE CLASS

Métodos	Descripciones
function build(path: String, C: Integer): DMTree	
	Creación de un objeto DMTree con capacidad C a partir de archivos distribuidos que contienen un conjunto vectorial.
function store(path: String, dmtree: DMTree)	
	Almacenar un objeto DMTree en archivos distribuidos.
function rebuild(path: String): DMTree	
	Reconstruir un objeto DMTree a partir de archivos distribuidos que contienen el objeto DMTree.
function kNNQuery(v: Vector, k: Integer): Set[Vector]	
	Ejecutar una consulta k-NN distribuida.
function rangeQuery(v: Vector, r: Double): Set[Vector]	
	Ejecutar una consulta de rango distribuido.

Fig. 2. El proceso de construir un objeto DMTree a partir de un conjunto vectorial.



Algoritmo 1. Creación de un objeto DMTree.

- Función: construir un objeto DMTree a partir de un conjunto vectorial en archivos distribuidos.
-Entrada:
1) path: la ruta de los archivos distribuidos que contienen el conjunto vectorial.
2) C: la capacidad del objeto DMTree.
- Salida: el objeto DMTree.

1. function build(path: String, C: Integer): DMTree{
2. cargue el conjunto vectorial de los archivos distribuidos en un objeto RDD[Vector].
3. crear un nuevo objeto DMTree vacío con capacidad C.
4. asignar cada parte del objeto RDD[Vector] a un objeto MTree en el objeto RDD[MTree] dentro del objeto DMTree.
5. devuelve el objeto DMTree.
6. }

B. Almacenamiento de un objeto DMTree en archivos distribuidos

Debido a que la creación de objetos DMTree lleva bastante tiempo, almacenaremos objetos DMTree para reutilizarlos más adelante. Un objeto DMTree puede ser muy grande, superando la capacidad de un archivo en un sistema de archivos local; por lo tanto, debe almacenarse en archivos distribuidos. El almacenamiento de un objeto DMTree en archivos distribuidos se describe en el Algoritmo 2.

Algoritmo 2. Almacenamiento de un objeto DMTree.

- Función: almacenar un objeto DMTree en archivos distribuidos.
- Entrada: 1) path: la ruta de los archivos distribuidos. 2) DMTtree: el objeto DMTree se almacenará en los archivos distribuidos.
- Salida: ninguna.

1. function store(path: String, dmtree: DMTree){
2. almacenar metadatos (la capacidad, el número de objetos MTree...) del objeto DMTree.
3. asignar cada objeto MTree en el objeto RDD[MTree] dentro del objeto DMTree a un archivo distribuido.
4.}

C. Reconstrucción de un objeto DMTree a partir de archivos distribuidos

Al reutilizar un objeto DMTree almacenado previamente, se puede reconstruir a partir de archivos distribuidos. Dado que un objeto DMTree incluye un conjunto de objetos MTree, cada ejecutor debe cargar al menos un objeto MTree para un procesamiento efectivo. Por lo tanto, el número de objetos MTree dentro de un objeto DMTree debe ser un múltiplo del número de ejecutores. El proceso de reconstrucción de un objeto DMTree a partir de archivos distribuidos en un clúster de Spark se describe en el algoritmo 3.

Algoritmo 3. Reconstrucción de un objeto DMTree.

-Función: reconstruir un objeto DMTree en un clúster Spark a partir de archivos distribuidos que contienen el objeto DMTree.
-Input: 1) path: la ruta de los archivos distribuidos que contienen el objeto DMTree.
-Output: el objeto DMTree.

1. function rebuild(path: String): DMTree{
2. cargar metadatos (la capacidad, el número de objetos MTree...) del objeto DMTree desde el archivo distribuido que contiene metadatos.
3. cree un nuevo objeto DMTree vacío con la capacidad cargada.
4. cargar objetos MTree de los archivos distribuidos en el objeto RDD[MTree] dentro del objeto DMTree.
5. devuelve el objeto DMTree.
6.}

D. Ejecución de una consulta k-NN distribuida

Una consulta k-NN distribuida es una consulta k-NN que se ejecuta en un clúster de Spark basado en un objeto DMTree. El proceso de ejecución de una consulta k-NN distribuida se muestra en la Fig. 3.

En primer lugar, el programa controlador en el nodo maestro invoca el método kNNQuery(v, k) en el objeto DMTree. Este método utiliza la transformación Map para asignar cada objeto MTree del objeto RDD[MTree] dentro del objeto DMTree a un objeto Set[Vector], que es el resultado de invocar el método kNNQuery(v, k) en el objeto MTree.

A continuación, el programa controlador recopila todos los objetos Set[Vector] de los nodos de trabajo al nodo maestro y los une al objeto Set[Vector] final mediante una acción Reducir. En Spark, la acción Reducir es una acción que recopila datos de un objeto RDD de los nodos de trabajo en el nodo maestro y realiza una función (como suma, min, max, unión...) en el conjunto de datos recopilados.

El objeto Set[Vector] final se ordena por distancias en orden ascendente. El resultado final es que los k vectores superiores se extraen del objeto Set[Vector] final.

El proceso de ejecución de una consulta k-NN distribuida se describe en el Algoritmo 4.

Algoritmo 4. Ejecución de una consulta k-NN distribuida.

-Función: ejecutar una consulta k-NN distribuida en un objeto DMTree.
-Entrada: 1) v: el vector de consulta. 2) k: el número de vecinos más cercanos.
-Salida: a lo sumo k vectores que están más cerca de v.

1. function kNNQuery(v: Vector, k: Integer): Set[Vector]{
2. asignar cada objeto MTree en el objeto RDD[MTree] dentro del objeto DMTree a un objeto Set[Vector], que es el resultado de invocar el método kNNQuery(v, r) en el objeto MTree.
3. recopilar todos los objetos Set[Vector] de los nodos de trabajo.
4. unión de todos los objetos Set[Vector] al objeto Set[Vector] final.
5. ordenar el objeto Set[Vector] final por distancias en orden ascendente.
6. devolver los k vectores superiores en el Set[Vector] final.
7. }

El problema de consulta k-NN distribuida se resuelve con éxito mediante el algoritmo 4, pero todavía hay un inconveniente que necesita más mejoras. Es decir, cada objeto MTree devuelve como máximo k vectores al nodo maestro, n objetos MTree devolverán como máximo (n × k) vectores al nodo maestro. Debido a que el nodo maestro solo extrae los vectores k superiores, hay muchos vectores recibidos por el nodo maestro, pero no utilizados. Esto aumenta el tráfico de datos transferido desde los nodos de trabajo al nodo maestro, disminuyendo el rendimiento del algoritmo. Sin embargo, superar esta debilidad no es fácil. Necesitamos investigar más en el futuro.

E. Ejecución de un range query distribuido.

Una consulta de rango distribuida es una consulta de rango

que se ejecuta en un clúster Spark basada en un objeto DMTree. El proceso de ejecución de una consulta de rango distribuida se muestra en la Fig. 4.

En primer lugar, el programa controlador en el nodo maestro invoca el método rangeQuery(v, r) en el objeto DMTree. Este método utiliza la transformación Map para

mapear cada objeto MTree en el objeto RDD[MTree] dentro del objeto DMTree a un objeto Set[Vector], que es el resultado de invocar el método rangeQuery(v, r) en el objeto MTree.

A continuación, el programa controlador recoge todos los objetos Set[Vector] desde los nodos trabajadores hasta el nodo maestro y los une al objeto Set[Vector] final mediante una acción Reduce.

El objeto Set[Vector] final se ordena por distancias en orden ascendente para facilitar su observación. El resultado final es el objeto Set[Vector] final.

El proceso de ejecución de una consulta de rango distribuido se describe en el Algoritmo 5.

Algoritmo 5. Ejecución de un range query distribuido.

- Función: ejecutar una consulta de rango distribuido en un objeto DMTree.
- Entrada: 1) v: el vector de consulta. 2) r: el radio.
- Salida: un conjunto de vectores tales que las distancias de v a ellos sean menores o iguales que r.

1. function rangeQuery(v: Vector, r: Double): Set[Vector]{
2. asignar cada objeto MTree del objeto RDD[MTree] dentro del objeto DMTree a un objeto Set[Vector], que es el resultado de invocar el método rangeQuery(v, r) en el objeto MTree.
3. recopilar todos los objetos Set[Vector] de los nodos de trabajo.
4. unión de todos los objetos Set[Vector] al objeto Set[Vector] final.
5. ordenar el objeto Set[Vector] final por distancias en orden ascendente.
6. devuelve el Set[Vector] final.
7. }

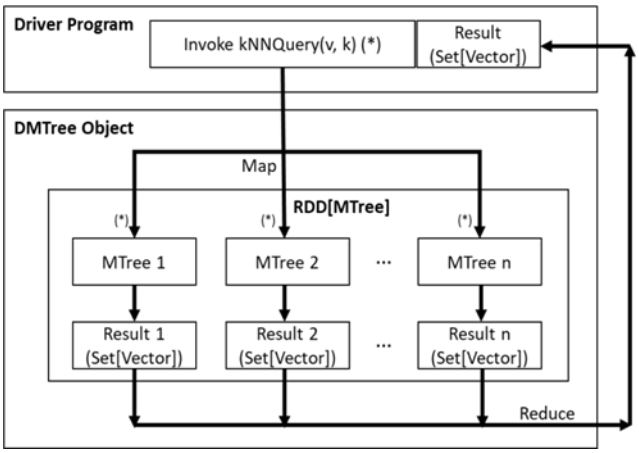


Fig. 3. El Proceso de Ejecutar una Distributed k-NN Query.

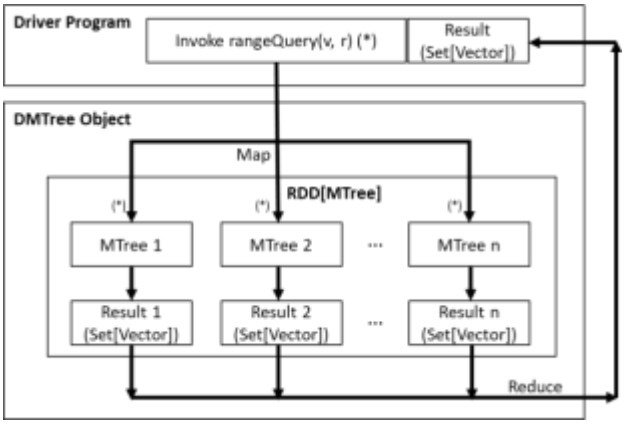


Fig. 4. El Proceso de Ejecutar una Range Query Distribuida.

IV. EXPERIMENTOS

En esta sección, mostramos los resultados experimentales de los objetos MTree y DMTree. Utilizamos la base de conocimientos Yago [16], descargada del sitio web del Instituto Max Planck de Informática [12], para crear los datos de los experimentos. En primer lugar, construimos grafos de conocimiento a partir de las tripletas descargadas y, a continuación, creamos conjuntos vectoriales de 64 dimensiones a partir de los grafos de conocimiento mediante la técnica de incrustación de grafos. A continuación se detallan los experimentos.

A. Experimentos con MTree Objects

Para realizar los experimentos con los objetos MTree, utilizamos un ordenador con la siguiente configuración:

- Procesador: Intel(R) Core™ i5-6500 CPU @ 3.20GHz 3.20GHz
- RAM: 16.0 GB

Realizamos experimentos con conjuntos de vectores de 64 dimensiones. Creamos objetos MTree con C = 1.000 a partir de estos conjuntos de vectores. La Tabla III muestra los resultados experimentales (en segundos) de 4 funciones:

- Construcción y almacenamiento de árboles MT: incluye la construcción de objetos MTree en la memoria del ordenador a partir de conjuntos de vectores almacenados en archivos locales y el almacenamiento de los objetos MTree en archivos locales para su reutilización posterior.
- Reconstrucción de árboles MT: reconstruye objetos MTree a partir de archivos locales.
- Ejecutar consultas k-NN: ejecuta consultas k-NN en objetos MTree.
- Ejecutar consultas de rango: ejecuta consultas de rango en objetos del árbol MT.

Sólo llevamos a cabo experimentos de hasta 2.000.000 de vectores porque si experimentamos con conjuntos de vectores más grandes excederán las capacidades de nuestro ordenador.

La Fig. 5 muestra el gráfico que compara el tiempo (en segundos) de construcción + almacenamiento y reconstrucción de objetos MTree. Este gráfico demuestra que construir y almacenar objetos MTree en archivos locales es más lento que reconstruir objetos MTree desde archivos locales a la memoria del ordenador. En concreto, construir y almacenar el objeto MTree a partir de un conjunto de vectores de 2 millones de vectores tarda 128,10 segundos, mientras que reconstruirlo tarda 76,86 segundos.

La Fig. 6 muestra el gráfico que compara el tiempo de ejecución de consultas k- NN y de rango en los objetos MTree. Este gráfico demuestra que la ejecución de consultas es bastante rápida y que las consultas k-NN son siempre más lentas que las consultas de rango. En concreto, la ejecución de una consulta k-NN en el objeto MTree de 2 millones de vectores tarda 7,82 segundos, mientras que la ejecución de una consulta de rango en el mismo objeto MTree sólo tarda 5,65 segundos.

B. Experimentos con DMTree Objects

Para realizar experimentos con objetos DMTree, hemos construido un cluster Spark que incluye 16 ordenadores. Uno de ellos es a la vez nodo maestro y nodo trabajador (para simplificar, nos referiremos a este ordenador como nodo maestro), y quince ordenadores funcionan únicamente como nodos de trabajo (Para simplificar, nos referiremos a estos ordenadores como nodos de trabajo). La configuración del nodo maestro es la siguiente:

- Procesador: Intel(R) Core™ i5-6500 CPU @ 3.20GHz3.20GHz
- RAM: 16.0 GB

Y la configuración de los nodos trabajadores como la siguiente:

- Procesador: Intel(R) Core™ i5-6500 CPU @ 3.20GHz3.20GHz
- RAM: 8.0 GB

TABLE III. RESULTADOS EXPERIMENTALES (EN SEGUNDOS) EN MTREE OBJECTS

Conjunto de Vectores	Creación y almacenamiento de MTrees	Reconstrucción de MTrees	Ejecución de consultas k- NN	Ejecución de Range Queries
0.1 m	56.68	34.01	0.26	0.27
0.2 m	58.09	34.85	0.53	0.48
0.3 m	60.52	36.31	0.99	0.76
0.4 m	65.13	40.68	1.34	1.02
0.5 m	69.98	44.39	1.71	1.34
0.6 m	75.09	45.05	1.88	1.63
0.7 m	77.78	46.67	2.36	1.87
0.8 m	81.06	48.64	2.50	2.16
0.9 m	86.91	52.14	2.85	2.46
1.0 m	91.84	55.10	3.19	2.81
1.1 m	97.35	58.41	3.84	3.00
1.2 m	102.28	61.37	3.86	3.29
1.3 m	104.05	63.83	4.57	3.63
1.4 m	110.41	66.24	4.62	3.90
1.5 m	112.59	67.56	5.61	4.19
1.6 m	114.30	68.58	5.57	4.62
1.7 m	116.53	69.92	5.76	4.83
1.8 m	118.34	71.00	6.22	5.23
1.9 m	123.27	73.96	6.62	5.54
2.0 m	128.10	76.86	7.82	5.65

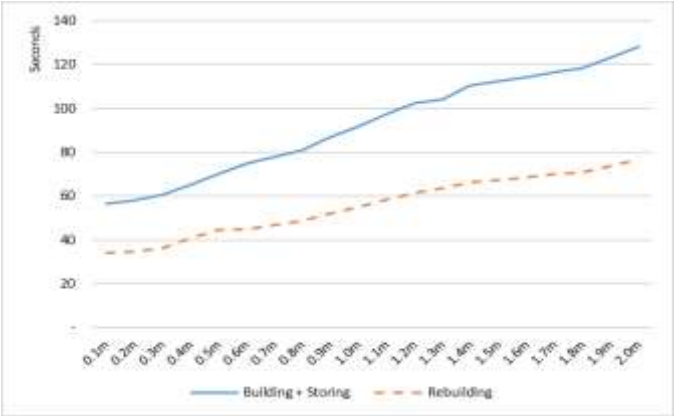


Fig. 5. Comparación del tiempo (en segundos) de construcción + almacenamiento y reconstrucción de objetos MTree.

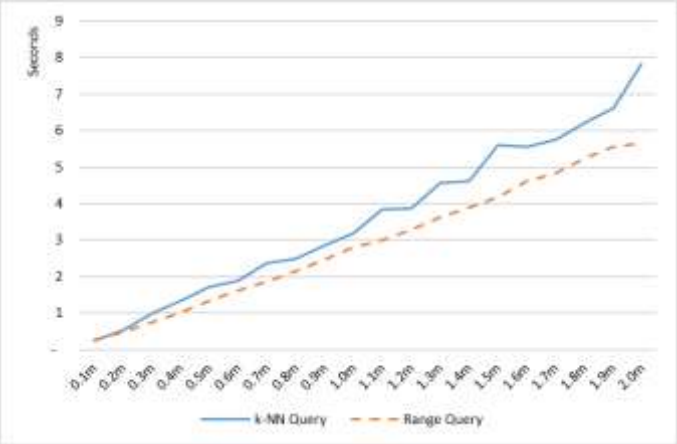


Fig. 6. Comparación del tiempo (en segundos) de ejecución de las consultas k-NN y Range en los objetos MTree.

La Fig. 7 muestra la arquitectura de nuestro cluster Spark. Donde:

- Nodo maestro: es un ordenador que ejecuta programas principales, envía código a los nodos trabajadores para que lo ejecuten en paralelo y recoge los resultados.
- Nodo trabajador: es un ordenador que participa en el procesamiento de las peticiones del nodo maestro.
- Gestor de clúster: es un componente que asigna recursos entre aplicaciones.El software instalado en nuestro clúster Spark se muestra en la Tabla IV.

De forma similar a los experimentos con objetos MTree, también realizamos experimentos con conjuntos de vectores de 64 dimensiones. Creamos objetos DMTree con C = 1.000 a partir de estos conjuntos de vectores. La tabla V muestra los resultados experimentales (en segundos) de cuatro funciones:

- Construcción y almacenamiento de DMTrees: incluye la construcción de objetos DMTree en el clúster Spark a partir de conjuntos de vectores almacenados en archivos distribuidos y el almacenamiento de los objetos DMTree en archivos distribuidos para su reutilización posterior.
- Reconstrucción de DMTrees: reconstruye objetos DMTree a partir de archivos distribuidos en el cluster de Spark.
- Ejecución de consultas k-NN distribuidas: ejecuta consultas k-NN en objetos DMTree.
- Ejecutar consultas de rango distribuidas: ejecuta consultas de rango en objetos DMTree.

La Fig. 8 es el gráfico que compara el tiempo (en segundos) de construcción + almacenamiento y reconstrucción de objetos DMTree. Este gráfico demuestra que construir y almacenar objetos DMTree es bastante lento. Sin embargo, cargar objetos DMTree desde HDFS al cluster Spark es mucho más rápido. En concreto, crear y almacenar el objeto DMTree a partir de un conjunto de vectores de 6 millones de vectores tarda 98,06 segundos, mientras que reconstruir este objeto DMTree sólo tarda 25,88 segundos.

La Fig. 9 es un gráfico que compara el tiempo de ejecución de consultas k-NN distribuidas y consultas de rango distribuidas en objetos DMTree. Este gráfico demuestra que la ejecución de las consultas es bastante rápido, y las consultas k-NN distribuidas son siempre más lentas que las consultas de rango distribuido. En concreto, ejecutar una consulta k-NN distribuida en el objeto DMTree de 6 millones de vectores tarda 6,48 segundos, mientras que ejecutar una consulta de rango distribuida en este objeto DMTree sólo tarda 5,19 segundos.

6,48 segundos, mientras que ejecutar una consulta de rango distribuido en este objeto DMTree sólo tarda 5,19 segundos.

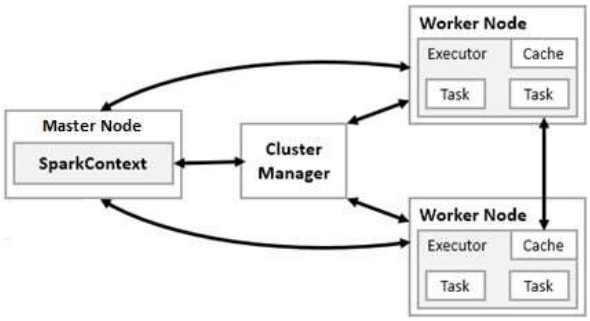


Fig. 7. Arquitectura de nuestro clúster Spark.

TABLE IV. SOFTWARE IS INSTALLED ON OUR SPARK CLUSTER

Software	Versión
Operating System	Ubuntu 18.04
Java	OpenJDK version 1.8.0_222
Scala	Version 2.11.12
Apache Hadoop	Apache Hadoop 2.8.5
Apache Spark	Apache Spark 2.4.4

TABLE V. RESULTADOS EXPERIMENTALES (EN SEGUNDOS) EN DMTREE OBJECTS

Conjunto de Vectores	Creación y almacenamiento de DMTrees	Reconstrucción de DMTrees	Ejecución de consultas k- NN distribuidas	Ejecución de consultas de rango distribuido (Range Queries)
1 m	18.38	2.66	2.11	1.27
2 m	21.92	5.59	2.84	1.65
3 m	28.25	9.09	3.72	2.74
4 m	41.21	12.00	4.77	3.22
5 m	69.95	18.25	5.12	3.85
6 m	98.06	25.88	6.48	5.19

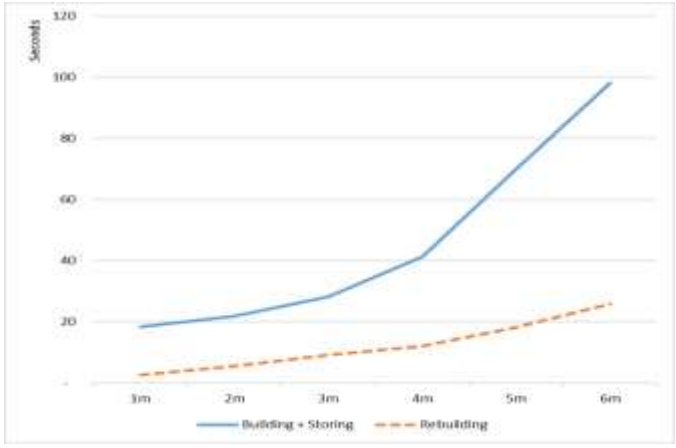


Fig. 8. Comparación del tiempo de creación + almacenamiento y carga de objetos DMTree.

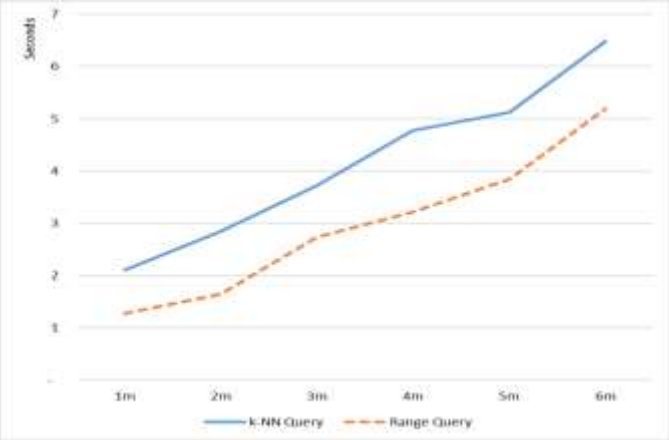


Fig. 9. Comparación de tiempos de ejecución de consultas distribuidas sobre objetos DMTree.

También es importante señalar que, si utilizamos objetos MTree en una máquina, sólo podemos indexar conjuntos de vectores de hasta 2 millones de vectores; pero si utilizamos objetos DMTree en el clúster Spark, podemos indexar conjuntos de vectores mucho mayores. Esto demuestra las limitaciones de la estructura MTree y la extraordinaria capacidad de la estructura DMTree.

V. Conclusiones y Trabajos Futuros

La estructura MTree es una técnica de indexación de conjuntos vectoriales. Podemos resolver eficazmente los problemas de búsqueda de similitudes (por ejemplo, k-NN y consultas de rango) en conjuntos de vectores utilizando la estructura MTree. Sin embargo, en el caso de grandes conjuntos de vectores distribuidos en varios ordenadores, la estructura MTree no es capaz de indexarlos. Para superar este inconveniente de la estructura MTree, ampliamos la estructura MTree para construir la estructura DMTree en el clúster Spark. También realizamos experimentos con ambas estructuras para demostrar que el rendimiento de la estructura DMTree es mejor que el de la estructura MTree.

A través del proceso de investigación, extraemos algunas ventajas y desventajas de la estructura DMTree como sigue.

Ventajas:

- Lleva tiempo crear la estructura DMTree una vez, pero se puede reutilizar muchas veces.
- Permite añadir o eliminar entradas a/de la estructura DMTree sin tener que reconstruirla.
- Ayuda a ejecutar eficazmente la consulta k-NN y la consulta de rango en grandes conjuntos de vectores.

Desventajas:

- La implementación de la estructura DMTree es bastante complicada.
- La estructura MTree contiene datos en nodos que hacen que el tamaño de la estructura MTree sea bastante grande, lo que resulta en un gran tamaño de la estructura DMTree.
- Encontrar similitudes en grandes conjuntos de vectores sigue siendo un poco lento debido al gran coste de comunicación entre el nodo maestro y los nodos trabajadores.

En un futuro próximo, seguiremos investigando para reducir la estructura DMTree y disminuir los costes de comunicación entre el nodo maestro y los nodos trabajadores para mejorar el rendimiento de la búsqueda de similitudes en grandes conjuntos de vectores.

Reconocimientos

Esta investigación está financiada por la Universidad Nacional de Vietnam Ciudad Ho Chi Minh (VNU-HCMC) con la subvención número DS2020-26-01.

Referencias

[1] P. Goyal and E. Ferrara, “Graph embedding techniques, applications, and performance: A survey,” Knowledge-Based Syst., 2018.

[2] H. Cai, V. W. Zheng, and K. C. C. Chang, “A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications,” IEEE Trans. Knowl. Data Eng., 2018.

[3] A. Grover and J. Leskovec, “Node2vec: Scalable feature learning for networks,” Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., vol. 13-17-Augu, pp. 855–864, 2016.

[4] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector set,” ArXiv, pp. 1–12, 2013.

[5] Q. Le and T. Mikolov, “Distributed representations of sentences and documents,” 31st Int. Conf. Mach. Learn. ICML 2014, vol. 4, pp. 2931–2939, 2014.

[6] P. Ciaccia, M. Patella, and P. Zezula, “MTree: An efficient access method for similarity search in metric spaces,” in Proceedings of the 23rd International Conference on Very Large Databases, VLDB 1997, 1997.

[7] T. Bozkaya and M. Ozsoyoglu, “Indexing Large Metric Spaces for Similarity Search Queries,” ACM Trans. Database Syst., 1999.

[8] A. Vlachou, C. Doulkeridis, and Y. Kotidis, “Peer-to-peer similarity search based on MTree indexing,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2010.

[9] Z. Kouahla, “Exploring intersection trees for indexing metric spaces,” in CEUR Workshop Proceedings, 2011.

[10] J. P. Bachmann, “The SuperMTree: Indexing metric spaces with sized objects,” ArXiv, pp. 1–14, 2019.

[11] P. Zezula, G. Amato, V. Dohnal, and M. Batko, “Similarity Search: The Metric Space Approach,” Springer. 2006.

[12] “YAGO Homepage.” [Online]. Available: https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago. [Accessed: 21-Jan-2019].

[13] “MTree Wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/MTree. [Accessed: 15-Mar-2019].

[14] P. Zecevic and M. Bonaci, Spark in Action. Manning Publications Co, 2017.

[15] R. Kienzler, Mastering Apache Spark 2.x : Scalable analytics faster than ever, vol. 22, no. S1. 2017.

[16] “YAGO (database) Wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/YAGO_(database). [Accessed: 21-Jan-2019].