"Año del Fortalecimiento de la Soberanía Nacional".


Escuela Profesional de Ciencia de la Computación

Estructura de Datos Avanzados

Laboratorio 06: M-Tree

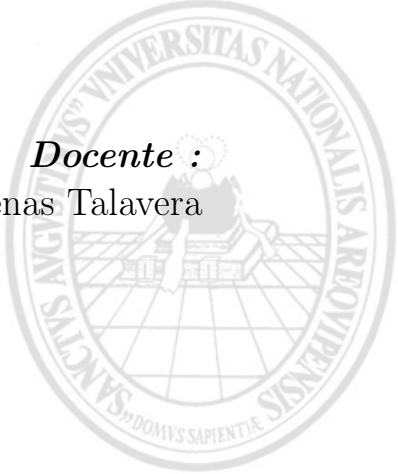
---

# DMTree: Un nuevo método de indexación para hallar similitudes en grandes conjuntos de vectores

---

***Alumnos:***
Jenny Huanca Anquise
Merisabel Ruelas Quenaya
Fiorela Villaroel Ramos

***Docente :***
Rolando J. Cardenas Talavera



# Índice

# 1. Alcances

Para solucionar los problemas a los que nos enfrentamos en el mundo real, usamos estructuras y algoritmos como el M-Tree. Los M-trees son estructuras de datos de árbol que son similares a los R-trees y B-trees . Se construye utilizando una métrica y se basa en la desigualdad del triángulo para consultas de rango eficiente y k-vecino más cercano (k-NN).

Una de las aplicaciones de M-Tree esta en la representación de millones o miles de millones de vértices en grafos de datos de todo tipo. Sin embargo, la aplicación de cálculos matemáticos y algoritmos de aprendizaje automático a los grafos es limitada y muy difícil. En situaciones concretas como la conversión de palabras en vectores para resolver problemas de Procesamiento del Lenguaje Natural se trabaja con la incrustación de grafo, donde cada vértice de un grafo se asigna a un vector de 64, 128, 256... dimensiones, cada dimensión es un número real. Para conservar toda la información posible del grafo original, el número de dimensiones del conjunto vectorial es mayor.

En la actualidad, son populares los conjuntos vectoriales de grandes dimensiones y las tareas de búsqueda de similitudes en estos conjuntos resulta en un coste computacional alto. Aunque M-Tree es una técnica para resolver los problemas de búsqueda de similitudes, solo se construyen en un ordenador. Es por eso que como alternativa se propone el desarrollo de una estructura optimizada en entornos de trabajo distribuido.

La mayoría de los trabajos anteriores no se refieren a la indexación de grandes conjuntos de vectores, donde los vectores están distribuidos en un clúster de ordenadores. Sin embargo, este trabajo propone encontrar similitudes de forma distribuida usando el framework Apache Spark, el famoso framework para el procesamiento distribuido, para construir la estructura DMTree para indexar grandes conjuntos de vectores de forma efectiva.

La distribución de este trabajo se divide en las siguientes secciones: Alcances, Funcionamiento que incluye, el problema, la solución planteada, conceptos relacionados, metodología y explicacion del algoritmo. También se presenta un análisis de la replicación, mejoras y finalmente las conclusiones.

# 2. Funcionamiento

## 2.1. Problema

Como se sabe la estructura Mtree es una técnica de indexación de conjuntos de vectores basada en una distancia definida , que puede resolver eficazmente los problemas de encontrar similitudes , sin embargo , dado que la estructura de M-Tree solo se construye en una computadora, solo puede indexar conjuntos de vectores pequeños, donde todos los vectores pueden almacenarse en una computadora. Hoy en día, los grandes conjuntos de vectores, donde los vectores se distribuyen en un grupo de computadoras, son cada vez más populares.

## 2.2. Solución

Para resolver este problema en [1] se propone un método para construir la estructura M-Tree distribuida (DM-Tree) extendido de la estructura M-Tree Spark un marco famoso

para el procesamiento distribuido , para indexar grandes conjuntos de vectores.

## 2.3. Conceptos

### 2.3.1. Definición de consulta de similitud

En conjuntos de vectores, la tarea común es encontrar similitudes. Generalmente se encuentran k vectores más cercanos al vector de consulta o encuentra todos los vectores en el rango del radio r, y el centro es el vector de consulta. Esas son la consulta de k-vecinos más cercanos (consulta k-NN) y la consulta de rango.

## 2.4. Metodología

- Primero, se realiza la implementación de la estructura M-Tree ejecutándose en una computadora.
- Después de eso, se extienda la estructura de M-Tree para la implementación de la estructura DM-Tree que se ejecuta en un clúster de Spark que consta de varias computadoras
- Finalmente se realizan muchos experimentos sobre ambas estructuras para probar que la estructura DM-Tree tiene una significativa mejora con respecto a la estructura M-Tree.

### 2.4.1. Construcción de un objeto DM-Tree

Como se menciono anteriormente en la metodología primero se realiza la implementación del M-Tree esta clase tiene los siguientes métodos importantes:

- **function insertObject (n: Node, v: Vector):** Esta función localiza el nodo hoja más adecuado en el subárbol del nodo n para almacenar un nuevo vector v. Es posible activar la división de la hoja nodo si el nodo hoja está lleno. Este método se utiliza para construir un objeto M-Tree a partir de un conjunto de vectores.
- **function kNNQuery (v: Vector, k: Integer): Set[Vector]:** Devuelve los k vectores que son más cercano a v
- **function rangeQuery (v: Vector, r: Double): Set[Vector]:** Devuelve todos los vectores tales que las distancias de v a ellos menores o iguales a r.

Una vez definido las funciones principales del M-Tree , veremos como se construye el DM-tree. El objeto DM-Tree incluye un conjunto de objetos de M-Tree que fuerón creados a partir de un conjunto de vectores. El conjunto de vectores se distribuye en particiones de un RDD (Resilencia de Conjunto de datos distribuidos), que es una estructura de datos fundamental de Apache Spark y es una colección tolerante a fallas de elementos que se puede operar en paralelo.

Como se observa en 2 explicaremos como es el proceso de construcción de un objeto DM-Tree a partir de un conjunto de vectores.

- Primero, se carga un conjunto de vectores desde archivos en un objeto RDD[Vector] en un clúster Spark .

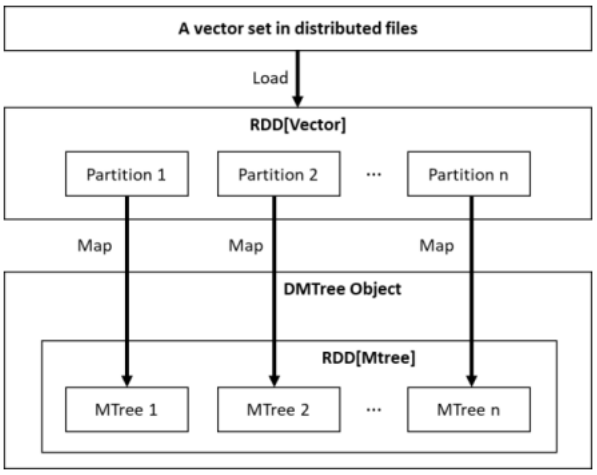



Figura 1: El proceso de creación de un objeto DM-Tree a partir de un conjunto de vectores [1]

- En segundo lugar, se crea un objeto DM-Tree.
- Luego, se usa la transformación *map* que transforma un objeto RDD[X] en otro objeto RDD[Y] en paralelo (se supone que X e Y son tipos de dato).
- Finalmente se asigna cada partición del objeto RDD[Vector] a un objeto M-Tree dentro del objeto DM-Tree.

## 2.5. Almacenamiento de un objeto DM-Tree en archivos distribuidos

Como la creación de objetos DM-Tree requiere de bastante tiempo, los objetos del DM-Tree se almacenaran para ser reutilizados más tarde. Un aspecto importante que se debe considerar es que puede ocurrir que se exceda la capacidad de un archivo en un sistema de archivos local, por esta razón se debe almacenarse en archivos distribuidos. El funcionamiento es el siguiente:

- Almacenar metadata(número de objetos M-Tree) del objeto DM-Tree.
- Mapear cada objeto M-Tree en el objeto RDD[M-Tree] dentro del Objeto DM-Tree a un archivo distribuido.

## 2.6. Reconstrucción de un objeto DM-Tree a partir de archivos distribuidos

Al reutilizar el objeto DM-Tree previamente almacenado se puede construir a partir de archivos distribuidos. Como el objeto DM-Tree incluye objetos M-Tree se requiere que cada ejecutor cargue al menos un objeto M-Tree para un procesamiento efectivo. Esto quiere decir que la cantidad de ejecutores debe ser un divisor de la cantidad de objetos M-Tree.

El funcionamiento es el siguiente:

- Primero se carga la metadata (número de objetos M-Tree) del objeto DM-Tree del archivo distribuido que contiene metadatos.
- Después se cree un nuevo objeto DM-Tree vacío con la capacidad cargada.
- Luego se cargan los objetos M-Tree de los archivos distribuidos en el Objeto RDD[M-Tree] dentro del objeto DM-Tree.
- Se retorna el objeto DM-Tree

## 2.7. Ejecución de una consulta k-NN distribuida

La diferencia entre un consulta K-NN y una distribuida es que esta última realiza la consulta en un cluster spark basado en un objeto DM-tree.

El funcionamiento es el siguiente:

- Primero, el programa controlador en el nodo maestro invoca el método kNNQuery(v, k) en el objeto DM-Tree , este método.
- Después se hace uso de la función map , que va ayudar a mapear cada objeto MTree en el objeto RDD[MTree] dentro del objeto DMTree.
- En cada objeto MTree se va a invocar el método kNNQuery(v, k) y este resultado se va a colocar en un objeto denominado set[vector]
- Una vez obtenido todos los resultados de cada MTree , se recopila todos los objetos Set[Vector] de los nodos trabajadores al nodo maestro y los une al objeto final Set[Vector] , mediante una acción reducir , en spark la función **reducir** es una acción que recopila datos de un objeto RDD de los nodos de trabajo en el nodo maestro y realiza una operación como (suma , unión , resta ..) en el conjunto de datos recopilados.
- Finalmente con el objeto final obtenido se ordena por distancias en orden ascendente , el resultado final es que los k vectores superiores se extraen del objeto Set[Vector] final.

## 2.8. Ejecución de una consulta por rango distribuida

Una consulta de rango distribuido es una consulta de rango que se ejecuta en un Spark cluster basado en un objeto DMTree.

Como se observa en la Fig 3 el funcionamiento es el siguiente:

- Primero, el programa controlador en el nodo maestro invoca el método kNNQuery(v, k) en el objeto DM-Tree , este método.
- Después se hace uso de la función map , que va ayudar a mapear cada objeto MTree en el objeto RDD[MTree] dentro del objeto DMTree.
- En cada objeto MTree se va a invocar el método rangeQuery(v, r) y este resultado se va a colocar en un objeto denominado set[vector]

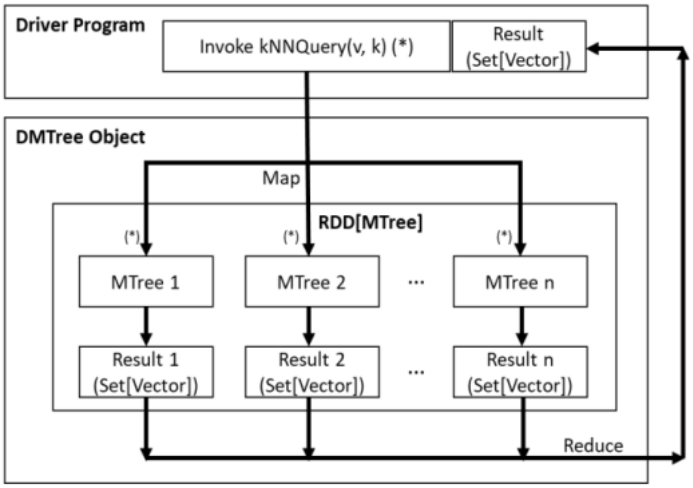



Figura 2: Ejecución de una consulta k-NN distribuida [1]

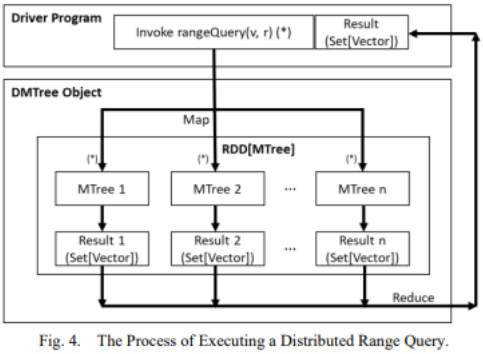



Fig. 4. The Process of Executing a Distributed Range Query.

Figura 3: Ejecución de una consulta por rango distribuida [1]

- Una vez obtenido todos los resultados de cada MTree , se recopila todos los objetos Set[Vector] de los nodos trabajadores al nodo maestro y los une al objeto final Set[Vector] , mediante una acción reducir , en spark la función **reducir** es una acción que recopila datos de un objeto RDD de los nodos de trabajo en el nodo maestro y realiza una operación como (suma , unión , resta ..) en el conjunto de datos recopilados.

- Finalmente con el objeto final obtenido se ordena por distancias en orden ascendente , el resultado final es que los k vectores superiores se extraen del objeto Set[Vector] final.

## 2.9. Experimentos

Para la realización de los siguientes experimentos se usó la base de conocimientos Yago [2] descargada del sitio web del Instituto Max Planck de Informátic[3]. Primero, se construyeron grafos de conocimiento a partir de las tripletas descargadas y, a continuación, se crearon conjuntos vectoriales de 64 dimensiones a partir de los grafos de conocimiento mediante la técnica de incrustación de grafos.

Los experimentos realizados fueron los siguientes:

### 2.9.1. Experimientos en Objetos M-Tree

Se realizaron con con conjuntos de vectores de 64 dimensiones. Luego se crearon objetos M-Tree con C = 1.000 a partir de estos conjuntos de vectores. El cuadro **??** muestra los resultados experimentales (en segundos) de 4 funciones:

- Construcción y almacenamiento de M-Trees: incluye la construcción de objetos M-Tree en la memoria del ordenador a partir de conjuntos de vectores almacenados en archivos locales y el almacenamiento de los objetos M-Tree en archivos locales para su reutilización posterior.
- Reconstruir M-Trees: reconstruye objetos M-Tree a partir de archivos locales.
- Ejecución de consultas k-NN: ejecuta consultas k-NN en objetos M-Tree.
- Ejecutar consultas de rango: ejecuta consultas de rango en objetos M-Tree.

| Conjuntos de vectores (mill) | Construir y almacenar M-Trees | Reconstrucción de M-Tree | Ejecución de consultas k-NN |
|---|---|---|---|
| 0.1 | 56.68 | 34.01 | 0.26 |
| 0.2 | 58.09 | 34.85 | 0.53 |
| 0.3 | 60.52 | 36.31 | 0.99 |
| 0.4 | 65.13 | 40.68 | 1.34 |
| 0.5 | 69.98 | 44.39 | 1.71 |
| 0.6 | 75.09 | 45.05 | 1.88 |
| 0.7 | 77.78 | 46.67 | 2.36 |
| 0.8 | 81.06 | 48.64 | 2.5 |
| 0.9 | 86.91 | 52.14 | 2.85 |
| 1 | 91.84 | 55.1 | 3.19 |
| 1.1 | 97.35 | 58.41 | 3.84 |
| 1.2 | 102.28 | 61.37 | 3.86 |
| 1.3 | 104.05 | 63.83 | 4.57 |
| 1.4 | 110.41 | 66.24 | 4.62 |
| 1.5 | 112.59 | 67.56 | 5.61 |
| 1.6 | 114.3 | 68.58 | 5.57 |
| 1.7 | 116.53 | 69.92 | 5.76 |
| 1.8 | 118.34 | 71 | 6.22 |
| 1.9 | 123.27 | 73.96 | 6.62 |

Cuadro 1: Resultados Experimentales en objetos M-Tree.

La figura 4 compara el tiempo (en segundos) de construcción + almacenamiento y reconstrucciónde 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 vectorial de 2 millones de vectores tarda 128,10 segundos, mientras que reconstruirlo tarda 76,86 segundos.

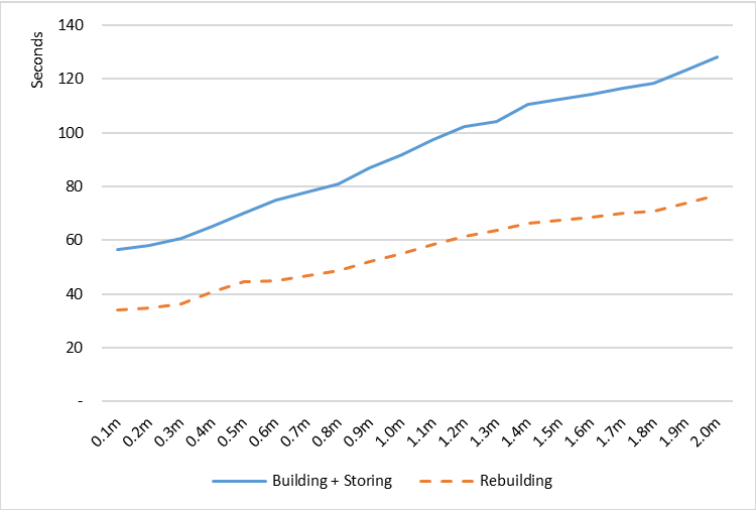



Figura 4: Comparación del tiempo (en segundos) de construcción + almacenamiento y reconstrucción de objetos M-Tree

La figura 5 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. 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.

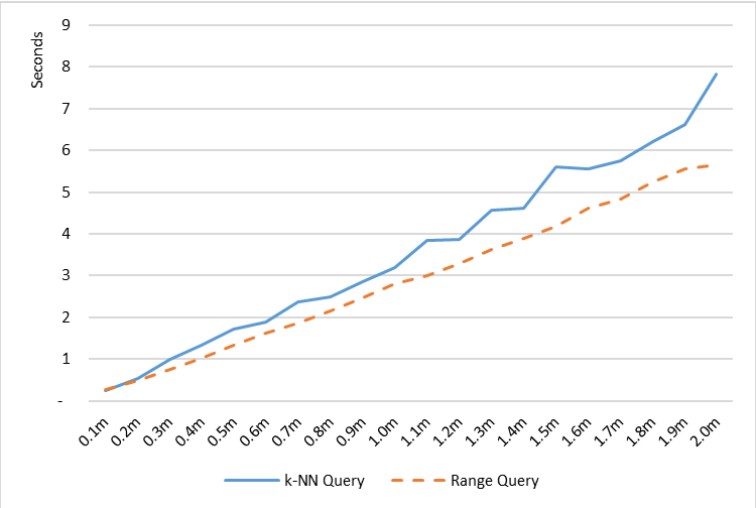



Figura 5: Comparación del tiempo (en segundos) de ejecución de las consultas k-NN y Range en los objetos M-Tree

### 2.9.2. Experimientos en Objetos DM-Tree

Para realizar experimentos con objetos DM-Tree, se construyó un cluster Spark que incluye 16 ordenadores. Uno de ellos es a la vez nodo maestro y nodo trabajador como se muestra en la figura 8. **Configuración de los nodos**

- 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.
  - Procesador: CPU Intel(R) Core™ i5-6500 a 3,20 GHz 3,20 GHz
  - RAM: 16,0 GB
- Nodos Trabajadores: Son ordenadores que participa en el procesamiento de las peticiones del nodo maestro.
  - Procesador: CPU Intel(R) Core™ i5-6500 a 3,20 GHz 3,20 GHz
  - RAM: 8,0 GB
- Cluster Manager: es un componente que asigna recursos entre aplicaciones.

De forma similar a los experimentos con objetos MTree, también se realizaron experimentos con conjuntos de vectores de 64 dimensiones. Se crearon objetos DMTree con C = 1.000 a partir de estos conjuntos de vectores. El cuadro 2 muestra los resultados experimentales (en segundos) de cuatro funciones:

- Creación y almacenamiento de DMTrees: incluye la creación 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 clúster Spark.
- Ejecución de consultas k-NN distribuidas: ejecuta consultas k-NN en objetos DMTree.
- Ejecución de consultas de rango distribuidas: ejecuta consultas de rango en objetos DMTree.

| Conjuntos de vectores (mill) | Construir y almacenar DM-Trees | Reconstrucción de DM-Tree | Ejecución de consultas k-NN distribuidas | Ejecución de consultas de rangos distribuidas |
|---|---|---|---|---|
| 1 | 56.68 | 34.01 | 0.26 | 0.27 |
| 2 | 21.92 | 5.59 | 2.84 | 1.65 |
| 3 | 28.25 | 9.09 | 3.72 | 2.74 |
| 4 | 41.21 | 12 | 4.77 | 3.22 |
| 5 | 69.95 | 18.25 | 5.12 | 3.85 |
| 6 | 98.06 | 25.88 | 6.48 | 5.19 |

Cuadro 2: Resultados Experimentales en objetos DM-Tree.

La figura 6 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

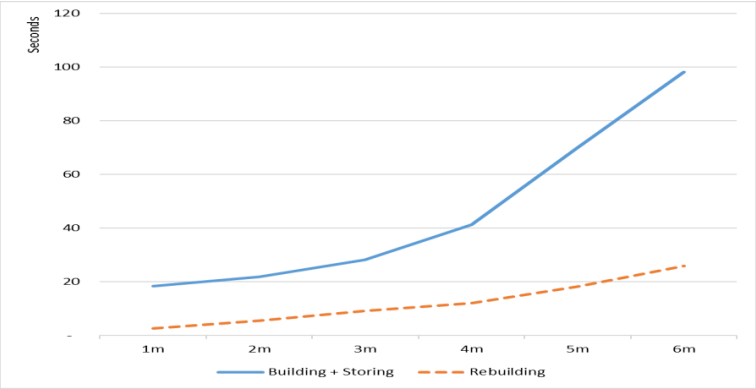



Figura 6: Comparación del tiempo de creación + almacenamiento y carga de objetos DM-Tree

DMTree a partir de un conjunto vectorial de 6 millones de vectores tarda 98,06 segundos, mientras que la reconstrucción de este objeto DMTree tarda Sólo 25,88 segundos.

La figura 7 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 lleva 6,48 segundos, mientras que ejecutar una consulta de rango distribuido en este objeto DMTree sólo tarda 5,19 segundos.

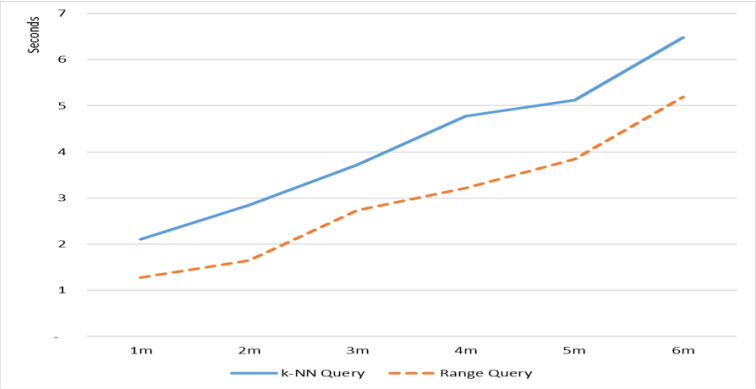



Figura 7: Comparación del tiempo de ejecución de consultas distribuidas en objetos DM-Tree

# 3. Capacidad de Replicación

Para replicar el DM-Tree se debe tener en cuenta las siguientes consideraciones:

- **Hadware:** Un computador que soporte el procesamiento de una gran cantidad de vectores. Como antecedente se tiene el trabajo con 2 000 000 de vectores en un computador con un procesador Intel(R) Core™ i5-6500 CPU @ 3.20GHz con 16 GB de RAM. En el caso de trabajar con archivos en un cluster de Spark el tamaño del conjunto de vectores puede ser más grande.

- **Conocmientos previos:**
  - Conocimientos sobre Apache Spark y su arquitectura. DM-Tree fue diseñado para ser usado en grandes conjuntos de vectores distribuidos en un clúster de ordenadores usando el framework Apache Spark pensado para el procesamiento distribuido.

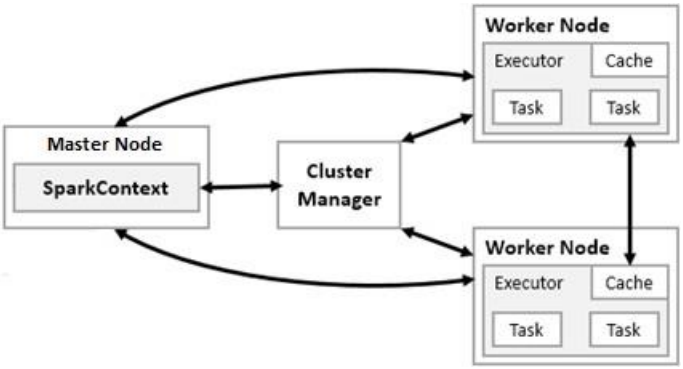



Figura 8: Arquitectura Spark Cluster usada

  - Conocimientos sobre M-Tree, una técnica de de indexación de conjuntos de vectores basada en una distancia definida. Utilizando la estructura MTree, se pueden resolver eficazmente los problemas de búsqueda de similitudes, que es el interés de este trabajo, pero aplicado a grandes conjuntos de vectores.
  - Conocimientos de software que pueda integrarse con Spark. En el caso de este trabajo se emplearon los siguientes softwares: Como sistema operativo, se uso Linux 18.04 junto a Java (OpenJDK version 1.8.0 222), Scala [4] (Version 2.11.12), Apache Hadoop 2.8.5 y Apache Spark 2.2.4.

| **Software** | **Versión** |
|---|---|
| Sistema operativo | Ubuntu 18.04 |
| Java | OpenJDK versión 1.8.0_222 |
| Scala | Versión 2.11.12 |
| Apache Hadoop | Apache Hadoop 2.8.5 |
| Apache Spark | Apache Spark 2.4.4 |

Figura 9: Software instalado en Spark Cluster

- **Implementación:** Para implementar DM-Tree dentro de un clúster de Apache Spark se puede usar como lenguajes de programación: Scala y/o Java (que fueron usados en el trabajo original), pero también puede usarse Python, R y lenguajes .NET.

# 4. Posibles Mejoras

A continuación se enumeran algunas mejoras que podrían hacerse a DM-Tree teniendo en cuenta las desventajas que presenta actualmente:

- La estructura de DM-Tree es complicada por lo que al momento de implementar los algoritmos presentados, se debe considerar el uso de un lengujae de programación que permita un manejo más sencillo para su mantenimiento y testeo. Una opción es el uso del lenguaje de programación Scala, que ofrece la ventaja de usar menos líneas de código, es usada ampliamente en tecnologías de Big Data y permite una infinidad de intregraciones, incluyendo Apache Spark.
- Debido a que DM-TRee es una estructura de gran tamaño la búsqueda de similitudes (KNN y consultas de rango) toma tiempo considerable por lo que se debería buscar maneras de reducir el tamaño de DM-TRee y disminuir el tiempo de comunicación entre el nodo maestro y los nodos trabajadores-

# 5. Conclusiones

La estructura M-Tree como técnica de indexación de conjuntos vectoriales resulta útil para resolver problemas de búsqueda de similitudes (por ejemplo, k-NN y consultas de rango) en conjuntos de vectores. Sin embargo, en el caso de grandes conjuntos de vectores distribuidos en varios ordenadores, la estructura MTree no es capaz de indexarlos. Por esa razón se propusó ampliar la estructura M-Tree para construir la estructura DM-Tree en el clúster Spark.

Se encontró algunas desventajas como:;

- La estructura de DM-TRee 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.

Sin embago, esta nueva implementación provee ventajas que la hacen útil para el manejor de un conjunto de vectores de gran tamaño:

- Una vez creada la estructura DM-Tree, se puede reutilizar muchas veces.
- La eliminación y adición de entradas no requiere la reconstrucción de DM-Tree.
- Es útil para resolver problemas de similitudes como KNN y consulta de rango en grandes conjutnos de datos.

Finalmente, como revisión de este trabajo, podemos concluir que una estructura como M-Tree puede ser optimizada sin hacer grandes cambios en su funcionamiento, solo aclopandola en entornos más optimizados como un framework de programación para procesamiento de datos distribuidos.

# Referencias

[1] P. Do, T. P. Hong, and H. D. To, "Dmtree: A novel indexing method for finding similarities in large vector sets," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 4, 2020.

[2] "Yago (base de datos) wikipedia."

[3] "Yago página de inicio."

[4] École Polytechnique Fédérale Lausanne, "The scala programming language."