ALGORITMO PARALELO BASADO EN EL MODELO FILTER-STREAM Y LA INFRAESTRUCTURA WATERSHED PARA BÚSQUEDAS POR SIMILITUD EN ESPACIOS MÉTRICOS

Norma Beatriz Perez, Mario Marcelo Berón
Universidad Nacional de San Luis, Facultad de Ciencias Físico Matemáticas y Naturales
Ejército de los Andes, 950 CP: D5700HHW - San Luis - Argentina
email: {nbperez, mberon}@unsl.edu.ar

Fernando Magno Quintão Pereira
Universidade Federal de Minas Gerais, Departamento de Ciência da Computação
Av. Antônio Carlos, 6627 - Prédio do ICEx - Pampulha - CEP: 31270-010
Belo Horizonte Minas Gerais - Brasil
email: fernando@dcc.ufmg.br

RESUMEN
Las búsquedas por similitud en los espacios métricos son de gran interés ya que permiten manipular tipos de datos complejos en un amplio conjunto de aplicaciones donde las técnicas tradicionales dejan de tener aplicabilidad. Por otro lado, existen aplicaciones que soportan una gran cantidad de datos. Al realizar el procesamiento de dichos datos los tiempos de resolución de una consulta resultan ser muy elevados. En estos casos, es necesario aplicar técnicas eficientes que permitan reducir los tiempos de búsqueda. En este sentido, la paralelización de estructuras de datos es un campo interesante de investigación. Una forma de abordar la problemática previamente mencionada, consiste en aprovechar los avances tanto del hardware como del software para elaborar algoritmos que realicen búsquedas eficientes, como lo son las búsquedas por similitud paralelas. Llevar a cabo la tarea antes mencionada es un proceso atrayente debido a que implica: i) El estudio de estructuras de datos que puedan ser paralelizables, ii) La selección de un modelo e infraestructura de computación paralela que facilite la tarea del programador y iii) La elaboración de algoritmos que utilicen eficientemente la estructura de datos elegida y que aprovechen al máximo las capacidades del modelo e infraestructura seleccionado. En este trabajo se describe la línea de investigación de paralelismo sobre conjuntos de datos métricos y un algoritmo de búsqueda por similitud basado en una estructura de indexación eficiente utilizando el modelo filter-stream asociado a una innovadora infraestructura denominada Watershed. Dicho algoritmo mostró un desempeño eficiente en la práctica respecto del número de procesadores disponibles, en redes de tamaño moderado.

Palabras Clave
Búsquedas por Similitud, Espacios Métricos, Modelos e Infraestructuras de Computación Paralela, filter-stream, Watershed, Paralelización.

INTRODUCCIÓN

La recuperación de la información sobre grandes conjuntos de datos tales como: registros de datos científicos, aplicaciones multimedia, entre otros, se han vuelto un problema de gran interés. Los tipos de datos que componen estas aplicaciones son, generalmente, complejos (datos semiestructurados, información biológica) por lo que el proceso de manipulación de ellos no es una tarea simple.

En muchos de esos casos las búsquedas exactas dejan de ser aplicables. Esto ha motivado el surgimiento de diversas técnicas y métodos que permiten construir estructuras de indexación. Estas estructuras permiten recuperar la información de manera más eficiente y en menos tiempo. Esté tipo de métodos se ubican en una categoría denominada *búsquedas por similitud* [1]. En esta categoría se ubican todos aquellos métodos que consisten en buscar aquellos objetos en un conjunto de datos que sean similares a un objeto de consulta dado. Las búsquedas por similitud pueden ser descritas a través de la definición formal de los *Espacios Métricos* [2]. Existen dos tipos de búsquedas típicas a saber:

Búsqueda por Rango:

Recupera los objetos que se encuentran a lo más a una distancia del objeto de consulta.

Búsqueda de K vecinos más cercanos:
Recupera los *k* objetos más similares al objeto de consulta dado.

Un Espacio Métrico esta compuesto por un par ordenado formado por un universo de objetos de un conjunto finito y de una *función de distancia* que satisface las propiedades de: *positividad*, *simetría* y *desigualdad triangular*. En donde, los objetos del universo serán comparados directamente utilizando la función de distancia, que indica el grado de *similitud* entre dos objetos.

El objetivo de los algoritmos de búsqueda consiste en minimizar la cantidad de *evaluaciones de distancia* realizadas al resolver consultas. Se ha comprobado que la implementación de algoritmos secuenciales de búsqueda por similitud no es apropiada debido a que los mismos no escalan bien para conjuntos de datos grandes, como por ejemplo: registros de bases de datos médicas, conjuntos de datos disponibles en redes sociales *on-line* y servicios Web. Por esta razón, las investigaciones recientes de nuestro equipo se han centrado en tres aspectos fundamentales:

- La búsqueda de estructuras de datos que sean eficientes e inherentemente paralelas.
- La selección de un modelo e infraestructura de computación paralela simple de entender y fácil de utilizar.
- La elaboración de algoritmos paralelos de búsqueda por similitud eficientes.

Las tres temáticas antes mencionadas se describen en la Sección Línea de Investigación.

METODOLOGÍA

La metodología empleada para el desarrollo de este trabajo se describe sucintamente a continuación:

- Se estudiaron diferentes estructuras de indexación relacionadas con las búsquedas por similitud con el propósito de seleccionar una estructura que sea paralelizable.
- Se analizaron diferentes modelos e infraestructuras de computación paralela, en la literatura, con la finalidad de seleccionar un modelo y una infraestructura asociada a dicho modelo, que presente un alto grado de abstracción para desarrollar algoritmos paralelos. Además, se estudio la existencia de algoritmos que aborden la problemática de la *búsquedas por similitud paralela*.
- Con los resultados obtenidos de los ítems anteriores se propuso un algoritmo paralelo que mejora el desempeño de los existentes respecto del tiempo de computación utilizado teniendo en cuenta el número de procesadores, o que sean innovadores en su ámbito.
- Se realizó un análisis experimental del algoritmo propuesto utilizando diferentes métricas que permitan verificar que provee un mejor desempeño con respecto a otros enfoques paralelos así como sobre el algoritmo secuencial.

LÍNEA DE INVESTIGACIÓN

La línea de investigación descrita en este artículo consta de tres temáticas principales las cuáles se describen a continuación.

1. Paralelización de Estructuras de Datos

En el estado del arte se pudo observar la existencia de un amplio conjunto de estructuras de datos [3] especializadas que permiten resolver búsquedas por similitud en los Espacios Métricos. Este conjunto se clasifica en dos grupos basados en:

Clustering:
Dividen el conjunto de datos en áreas. Cada área tiene un centro, y se intentan descartar áreas completas sólo comparando la consulta con el centro del área. Las estructuras: *M-trees* (MT) [4], *Spatial Approximation Tree* (SAT) [2] son ejemplos de este grupo.

Pivotes:
Almacenan las distancias que han sido calculadas previamente de cada objeto en el conjunto de datos a un conjunto de pivotes. Luego esas distancias se utilizan

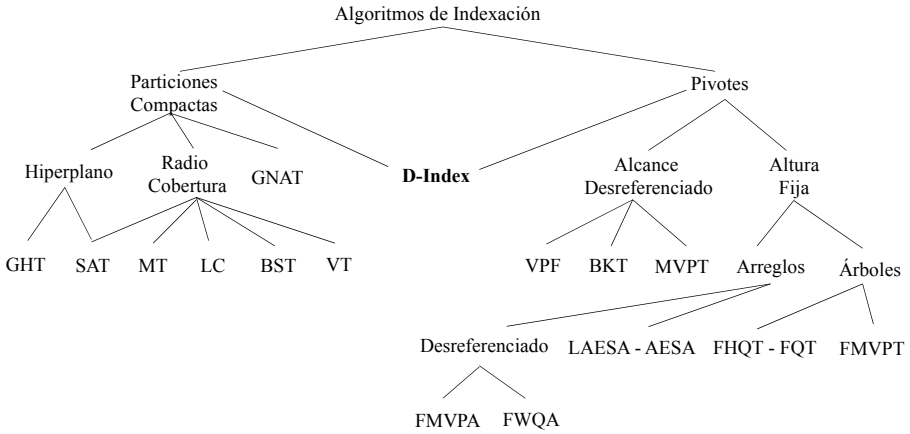

Figura 1: Taxonomía de los Algoritmo.

durante la búsqueda para descartar objetos del conjunto resultado. Las estructuras: *Approximating Eliminating Search Algorithm* (AESA) [5], *Fixed Queries trees* (FQT) [6] pertenecen a este grupo.

La Figura 1 ilustra una clasificación de las estructuras de indexación para búsquedas por similitud en los Espacios Métricos existentes en la literatura de acuerdo a sus características generales.

Luego de varios estudios realizados por el grupo de investigación se pudo concluir que el D-Index es una estructura apropiada para ser paralelizable. Esta afirmación se basa en las características del D-Index que se describen a continuación:

- El D-Index secuencial [7] es considerado uno de los métodos más rápidos de acceso disponible para resolver búsquedas por similitud [8].
- Representa uno de los avances más recientes en el escenario de los Espacios Métricos [3].
- Une la familia de algoritmos basadas en clustering y pivotes [3].
- Resulta adecuado en espacios de *alta dimensionalidad*, es decir, espacios en donde el problema de búsqueda por similitud es inherentemente difícil [8].
- Las características estructurales lo hacen adecuado para ser paralelizado.
- La performance depende de varios parámetros, y la configuración óptima de estos sigue siendo un problema abierto [8]. Esta situación abre la puertas para la realización de muchas investigaciones.

Debido a su importancia y con el propósito que el artículo quede auto contenido en la subsección siguiente se describe de manera sucinta la estructura de datos D-Index.

1.1 D-Index

El D-Index [7] es una estructura de indexación introducido por Dohnal en 2003. Se diseño para manejar grandes conjuntos de datos y, como tal, proporciona un buen tiempo de acceso a los datos almacenados en discos y otros medios de almacenamiento de alta latencia.

Sea $\mathbb{U}$ el universo de objetos y sea q un objeto de consulta, el D-Index permite la rápida localización de un pequeño subconjunto $\mathbb{X} \subseteq \mathbb{U}$ de tal manera que cada objeto $o \in \mathbb{X}$ se encuentra lo suficientemente cerca de la consulta q, de acuerdo a algún criterio de similitud.

El D-Index se basa en dos conceptos: una *función de distancia* que divide el universo de los objetos, y una *jerarquía de buckets* que almacenan estos objetos. A través de un ejemplo simple se describen los principios del D-Index. Para una descripción formal ver Dohnal [7].

El ejemplo consiste en buscar coordenadas en un espacio geométrico bi-dimensional. Se ha optado por este *espacio euclidiano*, ya que es intuitivo y fácil de representar en imágenes, sin embargo, el D-Index también se puede utilizar en otros tipos de Espacios Métricos.

La construcción del D-Index consiste en dividir sucesivamente el universo de los objetos en los diferentes *buckets* que conforman la estructura. Para esto se utiliza una función $s: \mathbb{U} \rightarrow \{0, 1, -\}$, que crea grupos o particiones. El proceso de construcción consta de dos etapas.

Primera Etapa: el proceso de partición del conjunto de datos se inicia eligiendo un objeto arbitrario o_1 del universo llamado *pivote*. A continuación se computa la función de *distancia mediana* (d_m) de cada objeto o_i con respecto al objeto escogido como pivote o_1. Esta situación se ilustra en la Figura 2 (a). Es importante destacar que el pivote o_1 y la función d_m son utilizadas para construir la función s de la siguiente manera:

$$s^{1,\rho}(o_i) = \begin{cases} 0 \; si \; d(o_1, o_i) \leq (d_m - \rho) \\ 1 \; si \; d(o_1, o_i) > (d_m + \rho) \\ - \; en \; otro \; caso \end{cases}$$

Segunda Etapa: una vez que se ha dividido el universo de los objetos en regiones separadas, el siguiente paso consiste en agrupar los objetos en los buckets. Esta situación se ilustra en la Figura 2 (b).

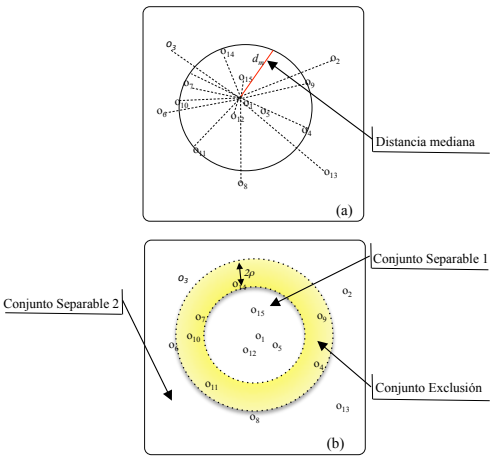

Figura 2: Ejemplo del D-Index.

Para esto el D-Index utiliza una jerarquía de buckets. Esta jerarquía esta compuesta por una serie de niveles. Cada nivel que conforma la estructura tiene 2^n buckets, donde n es el número de funciones que se utilizan para dividir el universo de los objetos. La configuración exacta de los buckets es un parámetro de la estructura de datos que debe ser fijado a priori.

El D-Index se puede construir con 1 o más niveles. La jerarquía de esta estructura se divide en dos partes principales: i) Cada nivel tiene asociado un conjunto de bukets separables (no necesariamente todos los niveles tienen la misma cantidad de bucket separables) y un bucket de exclusión temporario y ii) Un único bucket de exclusión global. La Figura 3 muestra una jerarquía de dos niveles, donde:

Primer Nivel:

Los objetos son extraídos del conjunto de datos. Cada objeto se intenta insertar en algún bucket separable de este nivel aplicando la función de partición asociada a este nivel. En el caso de que el objeto no se pudiera insertar en este nivel se almacena en el bucket de exclusión temporal de este nivel.

Segundo Nivel:

Los objetos son extraídos del bucket de exclusión temporario del nivel anterior. Cada uno de estos objetos se le aplica su función asociada para determinar en que bucket de ese nivel se debe almacenar dicho objeto.

Bucket de Exclusión:

Los objetos que se encuentren en el bucket de exclusión temporario del nivel anterior son almacenados finalmente en el bucket de exclusión global. Es importante notar que el bucket temporario puede no tener objetos almacenados.

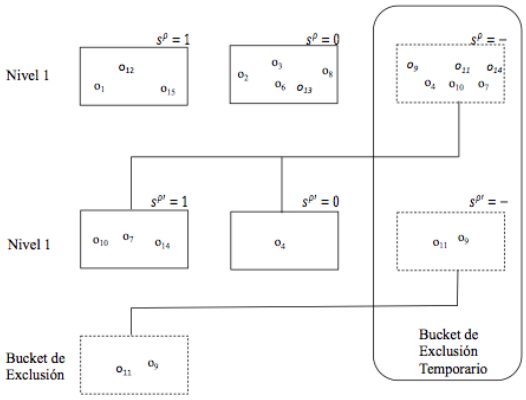

Figura 3: Arquitectura del D-Index.

2. Modelos e Infraestructuras de Computación Paralela

En la actualidad, la demanda de un mayor poder computacional a nivel científico, in-

dustrial, comercial, etc. se encuentra en continuo crecimiento. Esto se ve reflejado en aplicaciones de tiempo real, simulaciones, manipulación de grandes repositorios de información, entre otros. Como intento de proporcionar una solución a esta problemática han surgido una gran diversidad de arquitecturas robustas, redes de interconexión, técnicas de procesamiento paralelo, un amplio y diverso conjunto de modelos e infraestructuras de computación paralela, solo para citar algunos de los avances conseguidos. Sin embargo, el problema de diseñar e implementar algoritmos paralelos que sean eficientes, continúa siendo un interesante desafío. Esto motivó a los investigadores a desarrollar nuevos modelos e infraestructuras de computación paralela más eficientes siendo los mismos utilizados como guía en el diseño de los algoritmos, así como para estimar su rendimiento.

Los programadores se enfrentan continuamente al reto de utilizar un modelo e infraestructura de computación paralela que sea suficientemente detallado como para reflejar los aspectos realistas de la aplicación y que impacten en el rendimiento, al tiempo que sea suficientemente abstracto para ser independiente de la arquitectura y sencillo para el análisis.

Se hace evidente la necesidad de determinar un modelo e infraestructura de computación paralela que resulte adecuado para ser utilizado como soporte en la implementación de algoritmos paralelos. A continuación se describen de manera sucinta modelos e infraestructuras de computación paralela más usados por la comunidad científica dedicada al paralelismo.

- ***Bulk-Synchronous Parallel (BSP)***
  Modelo propuesto por Leslie Valiant [9]. BSP se define como un modelo de memoria distribuida con comunicación punto a punto entre los procesadores. Permite el desarrollo de software escalable e independiente de la arquitectura. Provee un entorno de trabajo simple y práctico para computaciones paralelas de propósito general.
- ***MapReduce***
  Es un modelo de programación desarrollado por *Google* [10] para el procesamiento y generación de grandes conjuntos de datos.
- ***DataCutter***
  Es una infraestructura [11] que permite la exploración y el análisis de conjuntos de datos científicos en ambientes distribuidos y heterogéneos.
- ***Anthill***
  Es una infraestructura [12, 13] que se basa en el modelo *filter-stream rotulado* [12, 14], que es una extensión del modelo de programación *filter-stream*.

Otros modelos [15] e infraestructuras [16, 17] de procesamiento que incluyen un conjunto variado de tecnologías y enfoques son propuestos en la literatura. Estos modelos sólo permiten resolver un subconjunto reducido, de manera eficiente, del amplio universo de aplicaciones existentes. Y si bien, ellos comparten algunos objetivos con los modelos e infraestructuras descritos previamente, presentan diferencias significativas que hacen que no sean considerados estándares. Estas diferencias se ven reflejadas en la arquitectura, modelo de datos, norma de comunicación, modelo de programación y garantías de disponibilidad y escalabilidad. Por lo que el uso y diseño de modelos e infraestructuras no es una tarea fácil e involucra un proceso altamente creativo.

Luego de haber estudiado diferentes modelos y sus infraestructuras asociadas se pudo observar que todos presentan sus ventajas y desventajas. No obstante, se seleccionó, en este trabajo, el modelo *filter-stream* por ser muy utilizado y porque se tiene interés en probar la nueva infraestructura Watershed. Esto se debe a que dicha infraestructura posee las siguientes ventajas:

– Es una infraestructura eficiente de reciente creación basado en el modelo *filtrer-stream.*

- Permite el diseño y programación de aplicaciones *on-line* y *off-line* distribuidas.
- Permite que las aplicaciones puedan ejecutarse: *Modo fluj*o (donde no existe el concepto de finalización) o *Modo por lotes* (donde los filtros finalizan tan pronto como sea procesado su último mensaje).
- Provee un mecanismo de adición y remoción de filtros en tiempo de ejecución.
- Permite la realización de alteraciones en la topología de las aplicaciones en tiempo de ejecución.
- Permite la reutilización de filtros en ejecución.

Debido a su importancia y con el propósito que este artículo quede auto contenido en la subsección siguiente se describe de manera sucinta el modelo asociado *filter-stream* y la infraestructura Watershed.

2.1 Filter-Stream

Tiene sus orígenes en trabajos como procesadores de *Data-Stream* de Duane Adams [18], y lenguajes de programación orientado al flujo, tal como Lucid [19]. Desde esas ideas pioneras, el modelo de programación *filter-stream* ha evolucionado sustancialmente.

Hoy en día, el flujo de procesamiento del hardware y los lenguajes son parte de la caja de herramientas de las compañías como IBM [20] y SPI [21], y actualmente es utilizado por haber sido el desarrollo económico de numerosas aplicaciones de gran tamaño [22]. Por otra parte este modelo, parece ser favorecido con un futuro prometedor, ya sea por computadoras *clouds* [23], procesadores *multi-core* [24] o unidades de *procesamiento gráfico* (GPU) [25, 26].

La característica principal del modelo *filter-stream* es la separación entre el procesamiento de datos y la comunicación de datos de manera modular. Los datos son agrupados en flujos, y luego enviados al ambiente de ejecución de los filtros que deberán procesarlos.

El modelo *filter-stream* permite dos visiones de la aplicación a saber:

Visión del Programador de la Aplicación:
Permite la abstracción del número de instancias de cada filtro que compone la aplicación en que se este trabajando. Es importante destacar que esta visión no tiene en cuenta donde los filtros serán ejecutados.

Visión de la Infraestructura:
Es utilizada por el usuario de la aplicación. En esta visión se debe tener en cuenta detalles como el número de instancias de cada filtro, el/los clusters donde los filtros serán ejecutados, entre otras consideraciones. Este nivel de detalle es muy importante para garantizar el desempeño adecuado de la aplicación.

2.2 Watershed

Es una infraestructura [27] de procesamiento distribuido asociada al modelo *filter-stream*. Se caracteriza por ser una infraestructura dinámica y flexible en la construcción de aplicaciones distribuidas. Ofrece abstracciones de alto nivel que posibilitan la exploración de ambos niveles de paralelismo, permitiendo el procesamiento *on-line* y *off-line* de grandes volúmenes de datos de forma eficiente.

Una aplicación en Watershed está compuesta por una cadena de elementos (*filtros*) de procesamiento. Los filtros se comunican por medio de flujos continuos de datos. Cada filtro representa una etapa del procesamiento realizado por la aplicación. Un flujo de datos está compuesto por resultados intermediarios de alguna etapa de procesamiento que son las entradas para etapas posteriores.

La Figura 4 muestra la arquitectura de Watershed y el procesamiento que realiza a través de filtros que intervienen en una aplicación arbitraria.

3. Algoritmos Paralelos

En la programación de algoritmos paralelos, el problema se divide en partes para que procesos separados ejecuten la computación de esas partes.
La construcción de algoritmos paralelos tie-

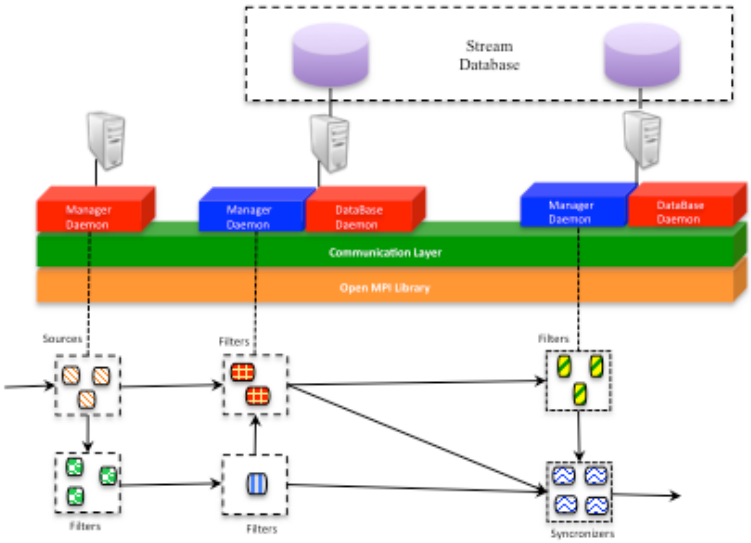

Figura 2: Arquitectura de Watershed.

nen como principal objetivo la resolución eficiente de instancias, que han sido asociadas a cada procesador, y que son parte de un problema complejo. Las técnicas para diseñar e implementar algoritmos paralelos aplican estrategias de descomposición o partición de datos, para dividir un problema en subproblemas de menor complejidad [28].

A continuación se describen de manera sucinta dos algoritmos paralelos relevantes encontrados en el estado del arte.

- ***Spatial Approximation Tree (SAT):***
  Se basa en la idea de aproximarse a la consulta espacialmente. Es decir, se comienza en algún punto en el espacio y se acerca cada vez más a la consulta. El SAT se construye de forma recursiva. Se selecciona un objeto *r* como la raíz del árbol y a éste se conectan todos los elementos en el conjunto de datos que están más cerca de *r* que a otro elemento. Los elementos que se conectan con *r* se conocen como los vecinos de *r*. El resto de los objetos se insertan en su vecino más cercano, de forma recursiva a medida que se van insertando en cada vecino. Para las búsquedas, se utiliza como criterio de descarte de objetos no relevantes el de los *hiperplanos* y el de *radio de cobertura*.
  El diseño algorítmico paralelo se basa en el uso del modelo de computación paralela *bulk-synchronous*. El lector interesado en conocer como funciona este algoritmo y los resultados que el produce puede leer [29].

- ***Índice de Distancia D-Index:***
  Combina una innovadora técnica de *clustering* con estimación de distancias basada en *pivotes* para acelerar la ejecución de consultas por similitud por rango y de vecinos más cercanos en repositorios de gran tamaño con objetos almacenados en memoria secundaria. A diferencia de las estructuras de árboles, el D-Index es adecuado para entornos dinámicos con una alta tasa de operaciones de eliminación/inserción.
  El diseño algorítmico paralelo de las técnicas propuestas se basa en el uso del modelo de computación paralela *bulk-synchronous*. Norma Perez et al. presentan en [30] la paralalelización de este algoritmo proponiendo diferentes técnicas paralelas para resolver búsquedas por rango de manera eficiente. Los resultados empíricos obtenidos, de un amplio y diverso conjuntos de datos, mostraron un desempeño eficiente en la práctica [30].

Otros algoritmos paralelos [31] son propuestos en la literatura que incluyen un

conjunto variado de enfoques. Sin embargo, es importante destacar que no existe un algoritmo que sea el óptimo, en todos los escenarios posibles, con respecto a los otros algoritmos existentes en el estado del arte. Por lo que el diseño de algoritmos paralelos es un desafío que involucra un proceso altamente creativo.

Ante la ausencia de un algoritmo óptimo para búsquedas por similitud el equipo de investigación propuso un algoritmo que mejora el desempeño de los algoritmos mencionados anteriormente. Dicho algoritmo utiliza el modelo *filter-stream*, la infraestructura Watershed y las características del D-Index para lograr desempeños relevantes a medida que se incrementa el número de procesadores.

RESULTADOS

A continuación se describe el algoritmo paralelo propuesto en este trabajo.

3.1 Algoritmo Propuesto

Los autores de este trabajo, proponer realizar la implementación del algoritmo descomponiendo el conjunto de datos en partes equitativas entre cada procesador que intervenga en el proceso de búsqueda. De esta manera cada procesador contiene un subconjunto diferente del conjunto de datos. Este enfoque tiene la ventaja de que cada procesador efectúa la búsqueda en una porción más pequeña del conjunto de datos acelerando los tiempo de respuesta. El algoritmo consta de dos partes claramente definidas. Cada una de ellas se describe a continuación:

3.1.1 Construcción del D-Index

El proceso de construcción del D-Index se describe a continuación:

Algoritmo de Construcción

1. Sea $\mathbb{X} \subseteq \mathbb{U}$, donde $\mathbb{X}$ es un conjunto de datos arbitrario y $\mathbb{U}$ representa el universo de objetos.
2. Sea $sizeBD = |\mathbb{X}|$, el tamaño del conjunto de datos.
3. Sea P un conjunto arbitrario de procesadores. $P = \{p_1, p_2, \dots, p_n\}$.
4. Sea $sizeProc = |P|$ la cantidad de procesadores que intervienen en una búsqueda.
5. Sea $sizeElem = sizeBD/sizeProc$ la cantidad de elementos que es asignada a cada procesador.
6. Para cada procesador p_i se construye el D-Index por medio de la invocación del algoritmo de inserción descrito en Donhal [8]. El tamaño de la estructura en cada procesador p_i es definido por el parámetro $sizeElem$ descrito en el punto 5. Los objetos son seleccionados del conjunto de datos $\mathbb{X}$ de manera aleatoria para su posterior inserción en el índice almacenado en el procesador p_i correspondiente.
7. Fin de la Construcción

La Figura 5 muestra el procesamiento de este algoritmo.

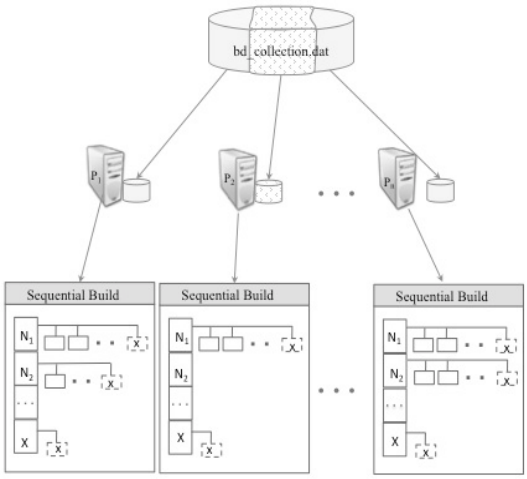

Figura 5: Procesamiento de Construcción.

3.1.2 Búsqueda en el D-Index

El proceso de búsqueda consta de:

Algoritmo de Búsqueda

1. Sea $\mathbb{Q} = \{q_1, q_2, \dots, q_m\}$, un conjunto arbitrario de consultas a realizar sobre la estructura implementada.
2. Sea $P = \{p_1, p_2, \dots, p_n\}$ un conjunto arbitrario de procesadores que intervienen en la búsqueda.
3. Para cada $q_i \in \mathbb{Q}$:
   3.1. Enviar la consulta q_i a todos los procesadores del conjunto P. De esta manera la misma consulta es realizada simultáneamente en una par-

tición diferente del conjunto de datos.
3.2. Invocar al algoritmo de búsqueda por rango del índice D-Index, descrito en Donhal [8].
3.3. Recolectar los resultados de cada procesador p_i.
3.4. Reportar los resultados.
4. Fin de la Búsqueda.

La Figura 6 muestra el procesamiento de este algoritmo de búsqueda.

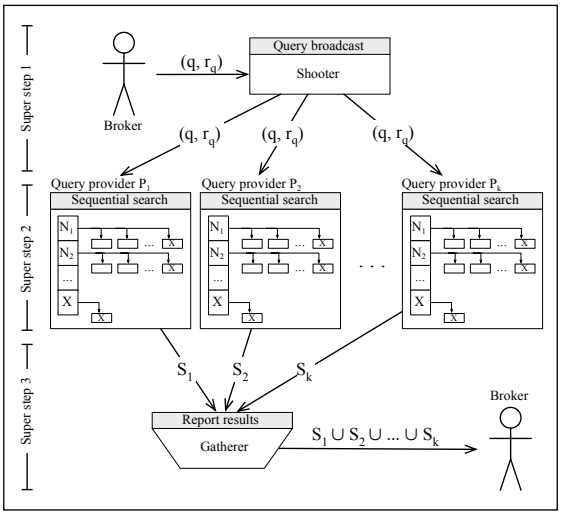

Figura 6: Procesamiento de Búsqueda sobre la Estructura Construida.

En la siguiente sección se presentan los resultados de los algoritmos descriptos.

Ambiente Experimental
Los experimentos realizados en este trabajo, se planificaron de acuerdo a las especificaciones que se detallan a continuación.

- Se realizaron ***búsquedas por rango***, tratando cada consulta con tres radios diferentes que cubren 0,01%, 0,1% y 1% del conjunto de datos.
- Base de datos utilizada: ***Imágenes de la Nasa***. Es un espacio de 40.150 vectores de dimensión 20, que representan vectores de características de imágenes tomadas de la NASA.
- Función de distancia utilizada: ***distancia Euclidiana L2***.
- El conjunto de datos se dividió en dos conjuntos aleatorios, el primer conjunto consta de un 90% de los datos que es utilizado para construir el D-Index, y el otro conjunto compuesto por el 10% restante de los datos que serán utilizados como consultas.
- La estructura D-Index ha sido construida con 32 niveles, con el siguiente número de pivotes {8, 7, 7, 7, 6, 6, 6, 6, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,3}. Es decir, que si se tienen 8 pivotes en el nivel 1 (2^8 buckets), 7 pivotes en el segundo nivel y así sucesivamente. Se eligió esta configuración porque los experimentos, que se han omitido en este trabajo, indican que se consiguen buenos tiempos de búsqueda.
- Las características del hardware usado para los experimentos es la siguiente:

Procesador	1x Intel Core 2 6420, 2 cores, 2 threads, 2.13GHz, 4MB de cache L2 (último nivel)
Memoria	2GB DDR2 667, en doble canal
Disco	SATA Seagate ST3250310AS, 250GB, 7200 rpm, 8 mb de cache
Placa madre	Intel Desktop Board DQ965GF
Red	Ethernet gigabit sem jumbo frame
Sistema operativo	Ubuntu Server 10.04 LTS, 64 bits

Tabla 1: Características del Hardware Utilizado.

Resultados Experimentales
Los resultados mostrados en la presente sección, se basan en las siguientes especificaciones.

- Se ha realizado 4.015 consultas con 3 radios diferentes, esto da un total de 12.045 consultas sobre el conjunto de imágenes de la NASA.
- Para los experimentos se utilizaron 2, 4, 6, 8 y 10 procesadores. También se uso 1 procesador, lo cual corresponde al caso secuencial. De esta manera se puede comparar las mejoras obtenidas al agregar paralelismo.
- Se utilizaron radios que recuperan el 0.1%, 0.01% y 1% del conjunto de datos para las consultas por rango. La Ta-

bla 1 muestra los valores de los radios obtenidos de manera experimental.

% Recuperación	Radio
0,01%	0,268387
0,1%	0,427567
1%	0,622274

Tabla 2: Radios Correspondiente a cada Porcentaje de Recuperación.

La Figura 7 muestra el tiempo promedio por consulta para diferentes radios de búsqueda. Se puede hacer la siguiente observación: a medida que se adicionan más procesadores para resolver una misma cantidad de consultas el tiempo de respuesta disminuye.

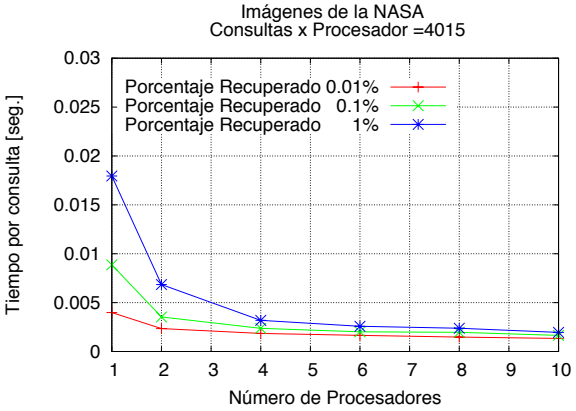

Figura 7: Tiempo Promedio por Consulta.

La Figura 8 muestra el *speedup* alcanzado experimentalmente para la colección imágenes de la Nasa. Se puede observar en el gráfico que el máximo *speedup* es obtenido cuando se aplica un radio de búsqueda cuyo porcentaje recupera el 1%. Esto se debe a que un radio de búsqueda mayor produce una mayor cantidad de trabajo sobre el conjunto de datos, por lo tanto el paralelismo es más efectivo en este caso.

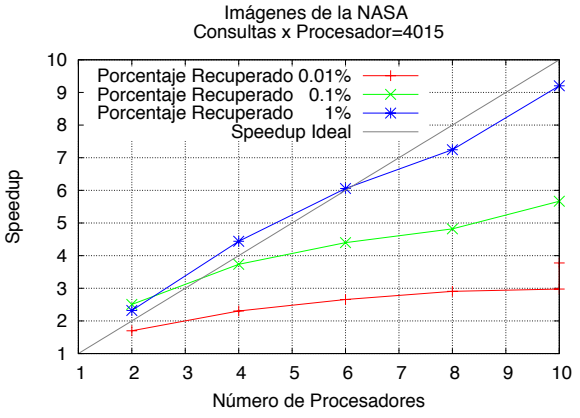

Figura 8: Speedup, del Conjunto de Datos Imágenes de la Nasa.

La Figura 9 muestra el *throughput* obtenido para diferentes radios de búsqueda. Puede ver que se logra un mayor *throughput* cuando se emplea un radio de búsqueda que recupera 0,01%. Esto se debe a que el radio es el más pequeño, lo que implica que se tienen que evaluar menos distancias que para los otros radios probados para este conjunto de datos, y por lo tanto mejora el rendimiento.

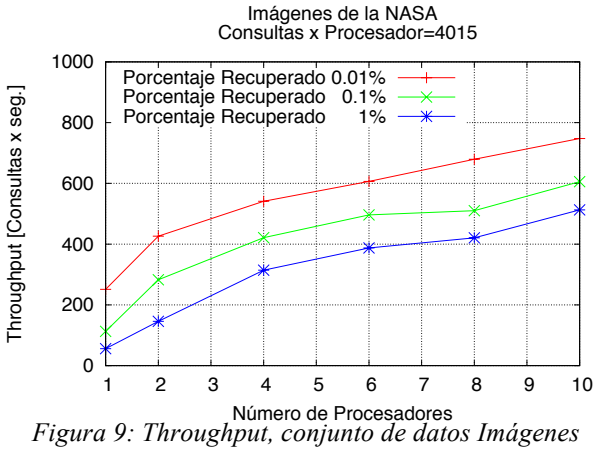

Figura 9: Throughput, conjunto de datos Imágenes de la Nasa.

DISCUSIÓN

El algoritmo paralelo propuesto en este trabajo es la primera implementación realizada utilizando la infraestructura Watershed la cual mostró una buen desempeño a medida que se adicionaron más procesadores. La aplicación de la infraestructura Watershed a este algoritmo puso en evidencia, a diferencia de otras infraestructuras en el estado del arte, que posee un alto grado de abstracción. Esto motiva a que cada vez más programadores de algoritmos paralelos lo escojan del amplio abanico de infraestructuras y modelos existentes en la literatura.

Por las razones mencionadas previamente, el equipo de esta línea de investigación se encuentra fuertemente comprometido a continuar explorando exhaustivamente la infraestructura Watershed; implementando nuevos enfoques paralelos y realizando una comparación exhaustiva entre esos enfoques a fin de determinar con cual de estos enfoques se consigue la máxima eficiencia. Además, determinar que condiciones son

necesarias para asegurar el máximo beneficio de ellos.

CONCLUSIÓN

La extraordinaria evolución en los últimos años de una gran variedad de tipos de datos, como por ejemplo datos multimedia y redes sociales en la Internet, hace necesario el estudio de nuevas estructuras de datos que permitan almacenar grandes cantidades de información, además de permitir su recuperación en forma rápida y eficiente.

Como resultado en este trabajo se seleccionó el D-Index como la estructura de datos más óptima y el modelo *filter-stream* con Watershed para implementar el soporte paralelo. Y ante la ausencia de algoritmos que escalen adecuadamente, en escenarios generales, para el problema que se intenta resolver se definió e implementó un algoritmo. El algoritmo paralelo propuesto que utiliza el modelo *filter-stream* y la infraestrucutra Watershed el cual posee características que hacen posible obtener resultados de desempeño interesantes. En este trabajo, en su análisis experimental, consigue alcanzar una escalabilidad próxima a la lineal (con radio = 0,622274), logrando un *speedup* cercano a 9.2 como se puede observar en la Figura 8. Para este análisis experimental se consideró los parámetros como: el número de pivotes, cantidad de niveles, cantidad de consultas y diferentes radios de búsqueda.

TRABAJOS FUTUROS

Los autores de esta línea de investigación proponen realizar como trabajo futuro:

- Experimentos sobre diferentes conjuntos de datos (donde se varían los tipos de datos, función de distancia, dimensión, cantidad de elementos, etc.). Esta tarea tiene como finalidad determinar que el algoritmo propuesto presenta un resultado similar con datos de diverso tipo no solo para búsquedas de imágenes. Esta tarea es relevante porque permitirá mostrar la generalidad del algoritmo.
- Implementación de nuevos algoritmos basados en diferentes enfoques paralelos y sobre otras infraestructuras o modelos de computación paralela para su posterior análisis y comparación con la versión implementada en Watershed.
- Realizar un análisis experimental de cada algoritmo paralelo utilizando métricas que permitan concluir la superioridad del algoritmo paralelo sobre el algoritmo secuencial.

REFERENCIAS

[1]. P. Zezula, G. Amato, V. Dohnal, and M. Batko, "*Similarity Search: The Metric Space Approach"*. 1st. ed. Springer Publishing Company, Incorporated, 2010.

[2]. E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín, "Searching in metric spaces". *ACM Comput. Surveys*., vol. 33, no. 3, pp. 273–321, 2001.

[3]. E. Chávez, J. L. Marroquín, and G. Navarro, "Fixed queries array: A fast and economical data structure for proximity searching", *Multimedia Tools Appl*., vol. 4, pp. 113–135, Jun. 2001.

[4]. P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces", *in VLDB. ACM*, pp. 426–435, 1997.

[5]. E. V. Ruiz, "An algorithm for finding nearest neighbours in (approximately) constant average time", *Pattern Rec*., vol. 4, pp. 145–157, 1986.

[6]. R. Baeza-Yates, W. Cunto, and S. Wu, "Proximity matching using fixed-queries trees", *In Proc. 5th Combinatorial Pattern Matching* (CPM'94), LNCS 807, pp. 198–212, 1994.

[7]. V. Dohnal, C. Gennaro, P. Savino, and P. Zezula, "D-Index: Distance searching index for metric data sets", *Multimedia Tools Appl*., vol. 21, no. 1, pp. 9–33, 2003.

[8]. A. J. Müller-Molina and T. Shinohara, "On the configuration of the similarity search data structure d-index for high dimensional objects", *in Proceedings of the 2010 international conference on Computational Science and Its Applications*, Volume Part III, ICCSA'10. Berlin, Heidelberg: Springer-Verlag, pp. 443–457, 2010.

[9].L. G. Valiant, "A bridging model for parallel computation" Commun. ACM, vol. 33, nro. 8, pp. 103–111, 1990.

[10]. J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters", *Commun*. ACM, vol. 51, pp. 107–113, 2008.

[11]. M. Spencer, R. Ferreira, M. Beynon, T. Kurc, U. Catalyurek, A. Sussman, and J. Saltz. Executing multiple pipelined data analysis operations in the grid. *In Proceedings of the ACM/IEEE conference on Supercomputing'02,* Los Alamitos, CA, USA. IEEE Computer Society Press, pp 1–18, 2002.

[12]. R. A. Ferreira, W. Meira Jr., D. Guedes, L. M. A. Drummond, B. Coutinho, G. Teodoro, T. Ta-

vares, R. Araujo, and G. T. Ferreira. Anthill: A scalable run-time environment for data mining applications. *In Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing*, SBAC-PAD'05, Washington, DC, USA. IEEE Computer Society, pp. 159–167, 2005.

[13]. D. Fireman, G. Teodoro, A. Cardoso, and R. Ferreira, "A reconfigurable run-time system for filter-stream applications", *in Proceedings of the 2008 20th International Symposium on Computer Architecture and High Performance Computing*, ser. SBAC-PAD'08. Washington, DC, USA: IEEE Computer Society, pp. 149–156, 2008.

[14]. A. Veloso, W. Meira, Jr., R. Ferreira, D. G. Neto, and S. Parthasarathy. Asynchronous and anticipatory filter-stream based parallel algorithm for frequent itemset mining. *In Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD'04, NY, USA. Springer-Verlag, pp. 422–433, 2004.

[15]. D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. V. Eicken. Logp: towards a realistic model of parallel computation. *In Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPOPP '93, NY, USA. ACM, pp. 1–12, 1993.

[16]. M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst*. Rev., 41(3):59–72, 2007.

[17]. C. Pu, K. Schwan, and J. Walpole. Infosphere project: system support for information flow applications. *SIGMOD Rec*., 30(1):25–34, 2001

[18]. D. A. Adams, "A computation model with data flow sequencing", *Ph.D. dissertation, Stanford University*, 1969.

[19]. E. A. Ashcroft and W. W. Wadge. Lucid, "a nonprocedural language with iteration". *Commun. ACM*, 20(7):519–526, July 1977.

[20]. B. Gedik, H. Andrade, Kun-Lung Wu, P. S. Yu, and M. Doo. "Spade: the system declarative stream processing engine". *In Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD'08, NY, USA, ACM, pp. 1123–1134, 2008.

[21]. B. K. Khailany, T. Williams, J. Lin, E. P. Long, M. Rygh, D. W. Tovey, and W. J. Dally. "A programmable 512 gops stream processor for signal, image, and video processing. Solid-State Circuits", *IEEE Journal*, 43(1):202 –213, jan. 2008.

[22]. W. Thies and S. Amarasinghe. "An empirical characterization of stream programs and its implications for language and compiler design". *In Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT'10, NY, USA, ACM, pp. 365–376, 2010.

[23]. D. G. Murray, M. Schwarzkopf, C. Smowton, S. Smith, A. Madhavapeddy, and S. Hand. "Ciel: a universal execution engine for distributed data-flow computing". *In Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, Berkeley, CA, USA, USENIX Association, pp. 9–9, 2011.

[24]. S. M. Farhad, Y. Ko, B. Burgstaller, and B. Scholz. "Orchestration by approximation: mapping stream programs onto multicore architectures". *SIGPLAN Not*., 46(3):357–368, 2011.

[25]. A. Hormati, M. Samadi, M. Woh, T. Mudge, and S. Mahlke. Sponge: Portable stream programming on graphics engines. *In ASPLOS'11*, 2011.

[26]. C. J. Rossbach, J. Currey, M. Silberstein, B. Ray, and E. Witchel. Ptask: "Operating system abstractions to manage gpus as compute devices*". In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP'11, NY, USA, ACM, pp. 233–248, 2011.

[27]. T. L. A. de Souza Ramos, R. S. Oliveira, A. P. de Carvalho, R. A. C. Ferreira, and W. M. Jr., "Watershed: A high performance distributed stream processing system", *in Proceedings of the 2011 23rd International Symposium on Computer Architecture and High Performance Computing*, ser. SBACPAD'11. Washington, DC, USA: IEEE Computer Society, pp. 191–198.

[28]. O. Gunther and H. Noltemeier. Spatial database indices for large extended objects. *In Proceedings of the Seventh International Conference on Data Engineering*, Kobe, Japan, pp. 520–526. IEEE Computer Society, april 8 -12, 1991.

[29]. M. Marín and N. Reyes "Efficient Parallelization of Spatial Approximation Trees." Computational Science, *Lecture Notes in Computer Science,* Vol. 3514, pp. 1003 – 1010, 2005.

[30]. N. B. Perez, N. Reyes and V. Gil-Costa "pD-Index: Un Índice Paralelo Eficiente para Búsquedas en Espacios Métricos". *Disertación tesis de Lic, Universidad Nacional de San Luis, 30 Julio, 2010.*

[31]. Q. Chu and L. Yongquan and G. Pengdong and W. Jintao and L. Rui "Parallel M-tree Based on Declustering Metric Objects using K-medoids Clustering", *in Proceedings of the 2010 Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science*, DCABES'10, IEEE Computer Society, Washington, DC, USA, pp. 206 -210.