Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

Tipo de artículo: Artículo original
Temática: Inteligencia Artificial
Recibido: 02/09/2016 | Aceptado: 06/02/2017

Algoritmo paralelo para la obtención de predicados difusos

Parallel Algorithm to obtain fuzzy predicates

Orenia Lapeira Mena [1*], Taymi Ceruto Cordovés [1], Alejandro Rosete Suárez [1], Humberto Díaz Pando [1]

[1] Facultad de Ingeniería Informática, Instituto Superior Politécnico "José Antonio Echeverría" (CUJAE), 114 #11901 e/ Ciclovía y Rotonda, Marianao, La Habana, Cuba. {olapeira, rosete, hdiazp}@ceis.cujae.edu.cu, taymiceruto@gmail.com

[*] Autor para correspondencia: olapeira@ceis.cujae.edu.cu

Resumen

El acelerado desarrollo en las distintas ramas de la ciencia y la ingeniería, exigen el diseño de novedosas técnicas de computación que permitan el procesamiento de grandes cantidades de datos, reduciendo los tiempos de respuesta y posibilitando el tratamiento de problemas complejos. FuzzyPred es un método de minería de datos que permite obtener predicados difusos en forma normal, como vía de representación del conocimiento. Para este método el tamaño de la base de datos es un factor esencial en los tiempos de respuesta del algoritmo, ya que cada predicado generado es evaluado en cada uno de los registros de la base de datos. Cuando este proceso se realiza de forma secuencial, se desaprovechan las arquitecturas de hardware que existen hoy en día para el procesamiento de grandes volúmenes de datos. Esto trae como consecuencia, que, en dependencia del tamaño de la base de datos, se pueden llegar a obtener largos tiempos de ejecución. En este trabajo se propone una versión paralela del algoritmo FuzzyPred, basado en la cantidad de datos que se pueden procesar dentro de cada uno de los hilos de procesamiento, de forma sincrónica e independiente. Los resultados obtenidos durante la experimentación realizada, indican que el algoritmo paralelo puede llegar a ser 10 veces más rápido que el secuencial y es por ello que se considera que puede ser muy útil en la mejora de la eficiencia del algoritmo ante bases de datos muy grandes.

Palabras clave: Paralelización de Datos, Predicados Difusos, Minería de Datos.

Abstract

The rapid development in several fields of science and engineering, requires the design of novel computational techniques that allow processing large amounts of data, reducing response times and enabling the treatment of complex problems. FuzzyPred is a data mining method that allows obtaining fuzzy predicates in normal forms. For

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

this method the size of the database is an essential factor in the response time of the algorithm, as each predicate is evaluated in each of the records in the database. When process is performed sequentially, it does not employ current hardware architectures that exist today for processing large volumes of data. This results in long runtimes, depending of the size of the database. This paper proposes a parallelized version of FuzzyPred, based on the amount of data that can be processed within each processing threads, synchronously and independent. The results obtained during experimentation indicate that the parallel algorithm is up to 10 times faster than the sequential version and that is why it is considered that can be very useful in improving the efficiency of the algorithm in very large databases.

Keywords*: Data parallelization, Fuzzy Predicates, Data Mining.*

Introducción

Los avances tecnológicos han permitido recopilar y almacenar grandes volúmenes de datos a lo largo de los años (Venugopal et al., 2009). Conjuntamente con esto, es significativo que las aplicaciones actuales posean un gran rendimiento y puedan analizar estos grandes conjuntos de datos de forma eficaz. Hoy en día, sigue siendo todo un reto para la minería de datos lograr que sus algoritmos y aplicaciones sean igual de eficientes ante el aumento del tamaño y dimensionalidad de los datos. Para lograr este objetivo muchas aplicaciones se apoyan en el paralelismo, debido a que es un área que permite la disminución del costo en función del tiempo de ejecución de los algoritmos ya que aprovecha las características de las arquitecturas de las computadoras actuales para ejecutar varios procesos de forma concurrente (Mostafa and Hesham, 2005; Hariri and Parashar, 2014).

FuzzyPred es un método de minería de datos, que permite la extracción de predicados difusos en forma normal conjuntiva y disyuntiva(Ceruto, 2014)(Ceruto et al., 2013). Este método está modelado como un problema de optimización combinatoria, debido a que el espacio de soluciones a recorrer puede llegar a ser muy grande. El algoritmo encargado de evaluarla calidad de cada predicado tiene una complejidad temporal polinomial de O (t*k*v), donde t es cantidad de registros, k es cantidad de cláusulas, y v es cantidad de variables) en el peor caso. Cada solución (o predicado) generado es evaluado en cada uno de los registros de la base de datos de forma secuencial. Teniendo en cuenta lo planteado anteriormente, y debido a que las dimensiones y la cantidad de variables de las bases de datos actuales, aumentan de tamaño todos los días, se pueden llegar a obtener altos tiempos de respuesta en este proceso por parte de FuzzyPred.

Debido a que la computación paralela deba ser aprovechada en la solución de problemas de minería de datos, en este artículo se presenta una versión paralela de FuzzyPred, con el propósito de reducir el tiempo de ejecución. El objetivo

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

fundamental del diseño empleado es aplicar un procesamiento paralelo enfocado en el tamaño de la base de datos, donde se aproveche, de forma flexible, las potencialidades de hardware que existen hoy en día. En el trabajo se realizan experimentos que permiten comparar la versión secuencial con la versión paralela de FuzzyPred, en diferentes métricas de rendimiento (particularmente, la aceleración y la eficiencia).

El trabajo está organizado de la siguiente forma. En la siguiente sección se exponen las principales temáticas del diseño de algoritmos paralelos, los paradigmas y herramientas de paralelismo que existen y sus características. Además, se hace una descripción de la biblioteca en la cual fue diseñado el modelo paralelo del algoritmo. Se establece, la temática de FuzzyPred como método de obtención de predicados difusos, donde se le da una especial atención al proceso de evaluación de los predicados en la base de datos y su costo computacional. Seguidamente, se explica la versión paralela del algoritmo FuzzyPred que se presenta y las bases de su diseño. Posteriormente se muestran los resultados experimentales obtenidos en varias bases de datos con diferentes características, comparando la versión secuencial de este algoritmo con la versión paralela en diferentes métricas de rendimiento para algoritmos paralelos. Finalmente, se muestran las conclusiones a las que se pudo arribar al concluir la etapa de experimentación.

Materiales y métodos

Diseño de Algoritmos Paralelos

Hoy en día muchos problemas necesitan procesar grandes cantidades de datos y hacer eficientes el tiempo de respuesta de cada una de estas aplicaciones. Para esto se cuenta con la eficiencia de un computador, el cual depende directamente del tiempo requerido para ejecutar una instrucción básica y del número de instrucciones que pueden ser ejecutadas al mismo tiempo (Ananth et al., 2003). La programación paralela es un área de la computación que permite aprovechar los recursos de hardware para mejorar el tiempo de ejecución de los algoritmos.

En la programación paralela existen dos tipos de paralelismo (Foster, 2003): paralelismo de control (descomposición funcional) o paralelismo de datos (descomposición de dominio). La descomposición de dominio o paralelismo de datos, como también se le conoce, consiste en una secuencia de instrucciones aplicadas a distintos datos. Los datos son divididos en partes y las partes son asignadas a diferentes procesadores. Cada procesador trabaja solo con la parte de los datos que son asignados, donde los procesadores pueden necesitar comunicarse para intercambiar los datos. El paralelismo de datos permite mantener un flujo único de control y seguir el modelo de Programa Único de Datos Múltiples (SMTP, por sus siglas en inglés).

En la descomposición funcional o paralelismo de tareas (también reparto dinámico de tareas) el problema se divide en un gran número de partes más pequeñas (muchas más partes que procesadores disponibles) y las sub-tareas son asignadas a procesadores disponibles. Tan pronto como un procesador termina una sub-tarea, realiza otra sub-tarea, hasta que terminen todas. El paralelismo de tareas se implementa sobre un paradigma de maestro y esclavos. El proceso maestro va asignando las tareas a los procesos esclavos, recogiendo los resultados producidos y asignando sub-tareas restantes (Foster, 2003). En los años recientes, debido al aumento de la escala del paralelismo que es necesaria en algunas situaciones, se ha llegado a acuñar el término Big Data, con su consiguiente marco conceptual y tecnológico(Fernandez et al., 2014).

Un algoritmo secuencial esencialmente sigue una secuencia de pasos para resolver un problema utilizando un solo procesador. Similarmente, un algoritmo paralelo sigue y soluciona la secuencia de pasos usando múltiples procesadores. Los algoritmos paralelos están diseñados de forma tal que se pueden resolver varios de estos pasos concurrentemente. Es esencial, para obtener cualquier beneficio del uso de computadoras paralelas, haber realizado un buen diseño del algoritmo.

En la práctica no es trivial realizar este diseño, por lo que se siguen un conjunto de pasos (se pueden incluir algunos o todos) para el diseño de un algoritmo paralelo, los cuales se exponen a continuación (Ananth et al., 2003; Foster, 2003):

1. Identificar las partes del algoritmo que son más costosas y que puedan ejecutarse de forma concurrente.
2. Mapear las partes que se pueden ejecutar concurrentemente dentro de múltiples procesos paralelos.
3. Distribuir los datos de entrada, salida e intermedios en el programa.
4. Permitir el acceso de los datos a múltiples procesadores.
5. Sincronizar los procesos de varios niveles en la ejecución del programa paralelo.

Paradigmas de la Programación Paralela

Para mejorar el tiempo de ejecución de un algoritmo se han creado dos paradigmas principales: paradigma de memoria compartida y el paradigma de memoria distribuida. El paradigma de memoria compartida consiste en que múltiples núcleos acceden a la misma memoria, al contrario de la memoria distribuida, donde se presentan varios núcleos, cada uno con su propia memoria local y que cooperan en conjunto para resolver tareas mediante alguna herramienta para administrar la ejecución de la tarea (Hoeger, 2011; Tosini,2011). A continuación, en la tabla 1, se abordan muy brevemente algunas características de ellas.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

Tabla 1: Características de los paradigmas para algoritmos paralelos

Paradigma	Ventajas	Desventajas
Memoria Compartida	Conceptualmente fáciles de programar Compartir los datos entre hilos de ejecución es muy rápido. **Se dispone de una infraestructura** (Para el caso de FuzzyPred).	La escalabilidad entre CPUs y memoria es mala. El programador es el responsable de la sincronía.
Memoria Distribuida	La escalabilidad entre CPUs y memoria es buena. Acceso rápido y exclusivo a la memoria.	El programador es el responsable de las comunicaciones. La conversión de programas secuenciales a paralelos no es un proceso trivial. La red de comunicaciones suele ser el cuello de botella. **No se dispone de una infraestructura** (Para el caso de FuzzyPred)

Para cada uno de estos paradigmas muchos investigadores han desarrollado conjunto de tecnologías que dan soporte a la paralelización, tanto en el paradigma en memoria compartida como en memoria distribuida. Estas herramientas ponen en práctica, todos los conceptos abordados anteriormente y apoyan el diseño de la programación paralela. Estas herramientas también conocidas como APIs (*Aplication Programing Interface*) facilitan el modelado de programas paralelos. A continuación, se abordan algunas características de las tecnologías que existen en la literatura hoy en día.

Algunas Tecnologías que implementan la programación paralela

Open MP (*Open Multi-Processing*)(Chapman et al., 2008; Pas, 2009): proporciona una API, que mediante directivas del compilador y llamadas a sub-rutinas, proporciona paralelismo de datos. La unidad base es el hilo y tiene acceso a variables en cache compartida o RAM. Los mejores resultados se ven cuando los accesos a los datos compartidos tienen bajo coste. Ofrece soporte en C, C++ y FORTRAN, aunque requiere de compiladores especiales (MP.org, 2014; Pas, 2009).

Intel TTB (*Intel Threading Building Blocks*): Modelo de programación paralela basado en rutinas que utilizan hilos. Provee una serie de plantillas, tipos de datos y algoritmos. TBB está pensado de cara al rendimiento, por lo que es compatible con otras bibliotecas y paquetes (Reinders, 2007).

Intel ArBB (*Inter Array Building Blocks*): biblioteca para C++ desarrollada para aprovechar diferentes tipos de procesadores en la resolución de problemas paralelos. Ofrece un modelo de programación basado en la composición dinámica de patrones estructurados de programación paralela. Son más sencillos de entender y depurar.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

JP: *Java Parallel* (PJ) es una API de programación paralela cuyos principales objetivos son: apoyar tanto a la programación paralela en memoria compartida (basadas en hilos de ejecución) y el grupo de programación paralela (basado en paso de mensajes), en una sola API unificada. Esta biblioteca les brinda a los desarrolladores la posibilidad de implementar programas paralelos que combinen ambos paradigmas. Es un software libre que se distribuye bajo los términos GPL (*General Public License*) y está implementado 100% en Java, específicamente en JDK 1.5. Las características de programación en memoria compartida están inspiradas en Open MP (Kaminsky, 2007; Kaminsky, 2015).

Map Reduce Hadoop: El marco de MapReduce es muy empleado en Big Data (Fernandez et al., 2014). Se basa en el hecho de que la mayoría de las tareas de procesamiento de información consideren una estructura similar, es decir, se aplica el mismo cálculo sobre un conjunto de máquinas. Los resultados intermedios de cada computadora se agregan de alguna forma. Esto sobretodo es útil cuando el volumen de datos es demasiado grande, lo cual es muy común en entornos como los asociados al manejo de información de las redes sociales. En este marco, es necesario que el programador especifique las funciones de mapa y reducción dentro de un trabajo. Luego, el trabajo divide el conjunto de datos de entrada en subconjuntos independientes que son procesados en paralelo por las tareas de mapa. MapReduce ordena las salidas de cada tarea y estas se convierten en entradas que serán procesadas por la tarea Reduce (Dean, 2008).

CUDA: (Dispositivo de Arquitectura Computacional Unificada o ***C****ompute* ***U****nified* ***D****evice* ***A****rchitecture*): es una biblioteca para implementaciones paralelas creada por la compañía NVIDIA y basada en el uso de GPUs (tarjetas de procesamiento gráfico) de dicha compañía. Se apoya en lenguajes conocidos como C, aunque para alcanzar un alto rendimiento es necesario realizar optimizaciones manuales sobre la configuración y amplio conocimiento de la arquitectura hardware de las GPU (Farber, 2011).

Métricas de Rendimiento

Cuando se cuenta ya con un algoritmo paralelo, también es necesario conocer el rendimiento que se alcanza al paralelizarlo, para de esta manera compararlo con la versión secuencial. A los criterios de evaluación de los algoritmos paralelos, también se les llama métricas de rendimiento o medidas de rendimiento. El criterio de evaluación puede ser simple o compuesto, si se combinan varias métricas (Hesham and Mostafa, 2005) . La evaluación del rendimiento de un algoritmo, se refiere a la medida de algún comportamiento del algoritmo como la exactitud-precisión, la robustez y adaptabilidad con diferentes complejidades. Medir el rendimiento de un algoritmo

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

paralelo permite conocer las características intrínsecas del algoritmo que deben ser enfatizadas, así como la evaluación de sus beneficios y limitaciones (Hesham and Mostafa, 2005).

Las métricas permiten explorar y refinar el diseño de un algoritmo paralelo, es decir, son modelos que pueden ser usados para un análisis cualitativo del rendimiento del algoritmo. Entre estas métricas más empleadas se encuentran: la aceleración o *speed up*, y la eficiencia.

La aceleración o *speed-up* es una medida que permite calcular el beneficio de un problema paralelo. Se define como la proporción del tiempo empleado secuencialmente con respecto al tiempo empleado en la versión paralela, aplicados a procesos idénticos(Ananth et al., 2003). Es decir, permite conocer cuántas veces es más rápido el programa paralelo con respecto al programa secuencial. Se denota por el símbolo S y se calcula de la manera siguiente, donde Ts representa el tiempo de ejecución secuencial y Tp el tiempo de ejecución paralelo.

$$S = \frac{Ts}{Tp}$$

La eficiencia es una medida que se brinda en función del tiempo. Está definida como la proporción de aumento de la aceleración (*speed-up*) entre el número de procesadores. Denota qué tan bien se han utilizado los procesadores, o sea, es la fracción de tiempo que los procesadores emplean para realizar las tareas asignadas(Ananth et al., 2003). Matemáticamente está definida por la siguiente ecuación, donde Ts es el tiempo de ejecución secuencial y Tp es el tiempo de ejecución paralelo y P el número de procesadores empleados.

$$E = \frac{Ts}{P \; x \; Tp}$$

El empleo de la computación paralela se convierte cada día en más grande y rápida, muchos problemas considerados anteriormente muy largos y costosos se han podido solucionar. El paralelismo se ha utilizado en muchos campos: estadística, medicina, meteorología, genética, geología, ingeniería eléctrica y mecánica(Beddo, 2002)(Fernandez et al., 2014)

La minería de datos y la programación paralela

La minería de datos abarca un campo interdisciplinario que tiene como principal objetivo extraer patrones que permitan identificar un conocimiento previamente desconocido desde las bases de datos(Han and Kamber, 2011). La tendencia al avance de la tecnología ha abierto las puertas a poseer enormes cantidades de datos, donde su análisis tomaría demasiado tiempo y costo realizarlo utilizando herramientas tradicionales, lo cual ha propiciado la aparición del término Big Data(Fernandez et al., 2014). Si al inicio el reto principal para la minería de datos era la obtención de

patrones que permitan obtener conocimiento, actualmente también lo es, el análisis de los grandes sistemas de datos con los que se cuentan hoy en día. Es por ello que el paralelismo resulta ser de gran importancia en esta área de la computación, como vía de solución para que los algoritmos y técnicas puedan procesar bases de datos con esta dimensionalidad.

Las tareas de la minería de datos pueden clasificarse en tareas descriptivas y predictivas. Los objetivos de las tareas de descripción es encontrar patrones y asociaciones interpretables en los datos. Por otra parte, las tareas de predicción consisten en encontrar posibles valores y distribución de atributos futuros (Hernández et al., 2004)(Han and Kamber, 2011; Kantardzic, 2011).

FuzzyPred

FuzzyPred es un método de minería de datos que propone los predicados difusos en forma normal conjuntiva y disyuntiva como vía de representación del conocimiento. Este método resuelve una tarea descriptiva donde no se conoce a ciencia cierta qué tipo de relaciones se van a encontrar, se trata de buscar patrones que describan los datos y sus relaciones. Este método está modelado como un problema de optimización combinatoria debido a que el espacio de soluciones por el que puede transitar puede llegar a ser inmenso(Ceruto T. , 2014).

Análisis de los procesos principales de FuzzyPred

Con el propósito de optimizar el tiempo de ejecución de FuzzyPred, fue realizado un estudio de los procesos principales, y que necesitan, desde el punto de vista computacional, más prestaciones, ya que poseen más carga de trabajo y se demoran en ejecutarse. Debido a sus características, los procesos identificados fueron: la evaluación de cada predicado y la etapa de post-procesamiento de los resultados.

El proceso de evaluación de los predicados interactúa con todo el sistema de datos. En este proceso resulta clave, tener en cuenta el tamaño que estos sistemas pudieran tener. Es importante en este proceso, recalcar que como consecuencia a toda la acumulación de información que se tiene actualmente, las bases de datos pueden llegar a ser inmensas, por lo que paralelizar este proceso es un aspecto fundamental para un buen funcionamiento de FuzzyPred, siempre que se ejecute en una computadora que tenga varios hilos de ejecución.

La etapa de post-procesamiento, por otra parte, tiene como principal objetivo ofrecer un conjunto de predicados más legibles para la comprensión del usuario y está formado por cuatro métodos principales: eliminar predicados repetidos, eliminar cláusulas iguales, decremento de variables y eliminar predicados evidentes. Estas funciones interactúan con todos los resultados obtenidos por FuzzyPred, y necesitan procesar la estructura de cada predicado, y

comparar, en la gran mayoría de los casos, con los restantes predicados del conjunto. Uno de los retos hoy en día de la minería de datos, es el gran número de soluciones que pueden estar brindando cada uno de los algoritmos. En el caso específico de FuzzyPred, el número de predicados obtenidos puede ser muy grande, aun cuando las bases de datos no son tan grandes, ya que el espacio de soluciones que puede estar recorriendo es enorme, debido a que la cantidad de variables del problema aumenta, siendo este último, un elemento significativo, porque se trabaja con etiquetas lingüísticas y no con los atributos reales de las bases de datos.

Para el caso de la etapa de post-procesamiento no es aplicable un modelo de paralelización funcional, debido que estas funciones son dependientes unas de otras. Fueron creadas con un orden definitivo e inviolable ya que los valores de salida de una representan los valores de entrada de la próxima función(Lapeira , 2012).

Para analizar cada uno de estos procesos, se tuvo en cuenta su complejidad algorítmica. La complejidad algorítmica(Guerequeta and Vallecillo, 1998)representa la cantidad de recursos temporales que necesita un algoritmo para resolver un problema y por tanto permite determinar la eficiencia de dicho algoritmo. Los criterios que se van a emplear para evaluar la complejidad algorítmica no proporcionan medidas absolutas sino medidas relativas al tamaño del problema.

En el proceso de evaluación existen ciclos anidados, el análisis de cada uno se va a realizar de adentro hacia afuera. El primer paso del algoritmo es evaluar cada variable de cada una de las cláusulas, este proceso tiene una complejidad de O (1) para cada variable, por lo que para todo el conjunto sería una complejidad O (v), donde v representa la cantidad de variables de una cláusula. El segundo proceso es evaluar cada cláusula de un predicado, para este proceso se tiene en cuenta si el predicado está en FNC o FND y la complejidad es O (k) * (O (v) + O (v)) = O (k) * max (O (v), O (v)) = O (k * v) donde k representa la cantidad de cláusulas. El tercer proceso es evaluar el predicado para cada registro de la base de datos, para este ciclo también se tiene en cuenta la estructura del predicado y la orden es O (t) * (O (k *v) + O (k)) = O (t) * max (O (k *v), O (k)) = O (t * k * v) donde t representa la cantidad de registros de la base de datos(Taymi, 2010).El tiempo de ejecución de una secuencia de instrucciones es igual a la suma de sus tiempos de ejecución individuales, lo cual es equivalente al máximo orden. En FuzzyPred es: max (O (t *k *v), O (t)) = O (t *k *v). Este tiempo de ejecución puede ser considerable ya que los factores que influyen en el mismo, también pueden tomar grandes valores. Es por ello que la siguiente sección se basa en la propuesta de paralelización de este algoritmo específicamente en el proceso de evaluación de los predicados difusos.

Propuesta de diseño paralelo en FuzzyPred como solución a la gran dimensionalidad de los datos

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

Para el diseño de paralelización de FuzzyPred específicamente en el proceso de evaluación se aplica el paradigma de paralelismo de datos, aplicado básicamente a la base de datos que se vaya a utilizar en el proceso de minería.
En este diseño se realizó la evaluación del predicado, en cada parte del sistema de datos de forma independiente y simultáneamente. Para esto se sigue un conjunto de pasos, los cuales se abordan a continuación:

1. En principio, se conoce la cantidad de hilos de ejecución del procesador de la computadora. Este proceso la biblioteca Java Parallel lo realiza de forma escalable.
2. Posteriormente, se procede a realizar grupos en dependencia de la cantidad de hilos que contenga la arquitectura donde se ejecute el algoritmo.
3. Consecutivamente, se asignan los grupos creados, los hilos de ejecución de forma dinámica buscando que todos los hilos tengan la misma cantidad de trabajo.
4. Para concluir se utilizó una barrera (*barrier*), para realizar la evaluación aplicando el cuantificador universal, ya que el mismo necesita conocer todos los valores de verdad del predicado en cada uno de los registros de la base de datos.

Posteriormente, en la Figura 1se muestra un diagrama de bloque donde se explica el diseño de paralelización para el proceso de evaluación de los predicados, identificando los parámetros de entrada, el diseño paralelo y las salidas del algoritmo. Este diagrama está basado solamente en el modelo de evaluación de los predicados difusos.

Debido a que no se dispone una infraestructura para el paradigma de memoria distribuida, es necesario en este trabajo emplear el paradigma de memoria compartida. Hoy en día la responsabilidad de conseguir un buen rendimiento no solo recae en el diseño, sino también en el software, porque los programas deben adaptarse para aprovechar las ventajas de las características paralelas que presenta el hardware.

La biblioteca seleccionada fue Java Parallel debido a que FuzzyPred se encuentra implementado en este mismo lenguaje y además Java Parallel soporta todos los paradigmas multi-núcleo de la programación paralela, incluyendo la memoria compartida y se adapta, a las características de la computadora en la que se ejecute, de forma escalable, siendo este último aspecto muy importante para el paradigma de memoria compartida.

Seguidamente, se expone el programa que contiene el diseño del modelo paralelo, utilizando las clases que provee la biblioteca Java Parallel para la modelación en memoria compartida.

En la versión paralela propuesta de la evaluación de los predicados, el programa crea un ***Parallel Team***, que es un objeto que contiene un número de hilos ocultos(Kaminsky, 2015). El número de hilos se puede especificar utilizando

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

el método *getThreadCount* (), de esta misma clase. Este método crea un hilo para cada CPU en la máquina SMP (Symmetric Multi-Processing). Entonces, el programa crea una sección paralela (***Parallel Section***) para cada hilo de ejecución y divide la cantidad de datos según la cantidad de hilos de ejecución, asignando paquetes de igual tamaño para ejecutar en la región paralela.

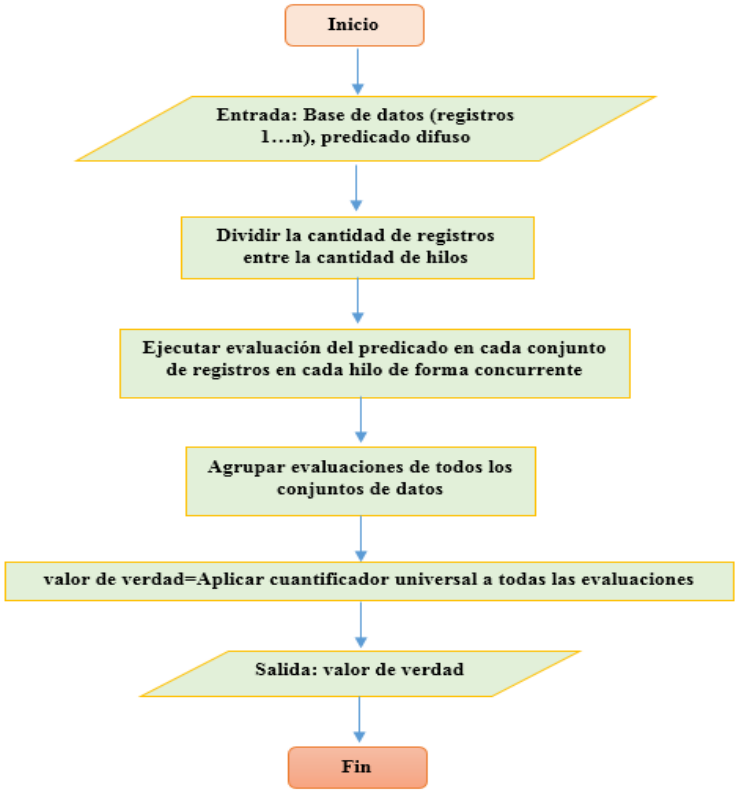

Figura 1: Diagrama en bloque del proceso paralelo de evaluación

Cada hilo del equipo invoca al método de ejecución de la región paralela (***Parallel Region***), y la máquina virtual de Java (JVM) ejecuta (***run***) simultáneamente en cada uno de los hilos, cada uno en su propio procesador. En cada iteración de cada sección (i), se ejecuta cada hilo. Este objeto proporciona un reparto de tareas en paralelo sobre el rango de índice de 0 hasta la cantidad de secciones. Por lo tanto, los hilos del equipo ejecutan simultáneamente diferentes porciones de los datos. Finalmente se utiliza el método addAll de java para agrupar todos los resultados de cada hilo de ejecución, para realizar el cuantificador universal.

Resultados y discusión

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

En esta sección se presenta la validación de la solución propuesta. Para esto se hace uso de las métricas de rendimiento aplicadas a algoritmos paralelos y se desarrolla una serie de pruebas comparativas. El objetivo de este acápite es comprobar que el tiempo de ejecución del algoritmo paralelo disminuye respecto a su versión secuencial.

Para los experimentos, se tuvieron en cuenta varias bases de datos, con diferentes características (se muestran en la Tabla 2). Estas bases de datos, son del entorno real y fueron tomadas de la *UC Irvine Machine Learning Repository*, la cual pone a disposición de los investigadores, una amplia gama de datos recopilados de diferentes áreas. Las bases de datos escogidas poseen diferentes tamaños, con el propósito de valorar esta característica.

Tabla 2: Descripción de las bases de datos empleadas en la experimentación

Nombres	**# Registros**	**# Atrib (R/N/E/)**	**# Etiquetas Lingüísticas**
Quacke	2178	4 (3/1/0)	9 (9/0/0)
Stulong	1417	5 (5/0/0)	15 (15/0/0)
Bolts	40	8 (2/6/0)	18 (18/0/0)

El algoritmo fue probado en Java bajo el ambiente de desarrollo Eclipse, compilados con el JDK 1.7. Para analizar el comportamiento del algoritmo paralelo, fueron diseñados tres escenarios, con diferentes objetivos. El objetivo del primer escenario es comparar la versión secuencial con la versión paralela de FuzzyPred, para esto ambas versiones fueron ejecutadas en la misma computadora con iguales prestaciones de hardware y bajo la misma configuración de entrada del FuzzyPred (los mismos parámetros de entrada del algoritmo y en una misma base de datos). Estos parámetros se encuentran reflejados en la Tabla 3.

Tabla 3: Parámetros de configuración del escenario 1

Parámetros de FuzzyPred	*operador de lógica difusa=zadeh, conectiva=aleatoria, escala de valores=0-12, ejecuciones=50, iteraciones por ejecución=1000, objetivo=valor de verdad, algoritmo metaheurístico=búsqueda aleatoria, base de datos=Quacke.*
Características de la PC	Intel Core i3 -2100 CPU 4 Gb de RAM

Tal y como se muestra en la tabla 4 se tomó el tiempo de ejecución (en minutos) de FuzzyPred en ambas versiones (secuencial y paralelo). Es importante aclarar que a pesar que las bases de datos no poseen una gran cantidad de registros (lo que representa un factor negativo ya que pudiera ser que la propuesta paralela no muestre mejora) el tiempo de ejecución paralelo mejora al tiempo de ejecución secuencial en un 10%, demostrando para esta última versión una mejoría.

Los resultados alcanzados en este experimento son los siguientes:

Tabla 4: Tiempos de ejecución de la versión secuencial vs paralela

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

Métrica de Calidad		Valor
Tiempo de ejecución	tiempo secuencial	87 min
	tiempo paralelo	79 min
Speed-up		1.10
Eficiencia		0.12

El objetivo del segundo escenario es comparar la versión paralela de FuzzyPred frente a varias computadoras con diferentes características y poder conocer el comportamiento del mismo en diferentes entornos de hardware. A continuación, se muestra en la tabla5 los parámetros de configuración de FuzzyPred y en la tabla 6 las características de cada una de las computadoras en las que se ejecutan los experimentos.

Tabla 5: Parámetros de configuración del escenario 2

Parámetros de FuzzyPred	*operador de lógica difusa=zadeh, conectiva=aleatoria, escala de valores=0-12, ejecuciones=50, iteraciones por ejecución=1000, objetivo=valor de verdad, algoritmo metaheurístico=búsqueda aleatoria, base de datos=Quacke.*

Tabla 6: Características del hardware aplicado en el escenario 2

Hardware	Memoria		
	Cantidad de procesadores	Tipo	Cantidad (Mb)
Intel Core 2 Duo E7300	2 núcleos	DDR2	2048
Intel (R) Core TM 2 Quad Q9300	4 núcleos	DDR2	4096
Intel Core i7 920	8 núcleos	DDR 3	2670

Posteriormente, en la tabla 7 se muestran los resultados alcanzados en cada una de las arquitecturas de hardware, respecto al tiempo de ejecución paralelo, el tiempo de ejecución secuencial, y los valores para las métricas de aceleración y eficiencia.

Tabla 7: Resultados obtenidos de la versión paralela en diferentes hardware.

Hardware	T. Secuencial	T.Paralelo	Aceleración	Eficiencia
Intel Core 2 Duo	129 min	98 min	1.31	0.66
Intel (R) Core TM 2 Quad Q 9300	96 min	62 min	1.54	0.39
Intel Core i7 920	73 min	40 min	1.82	0.30

Los resultados arrojados en el escenario 2, reflejados en la figura 2, respecto al tiempo de ejecución, demuestran que los mejores valores son encontrados en la arquitectura Intel Core i7, debido a que es el que mejores prestaciones de computo posee. Sin embargo, en la figura 3 se muestran los resultados en cada una de las medidas tenidas en cuenta en este trabajo, donde se puede observar que la aceleración aumenta a medida que va mejorando las características del hardware, por el contrario de la eficiencia, la cual disminuye, pues nuestro diseño es capaz de explotar las características del hardware.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

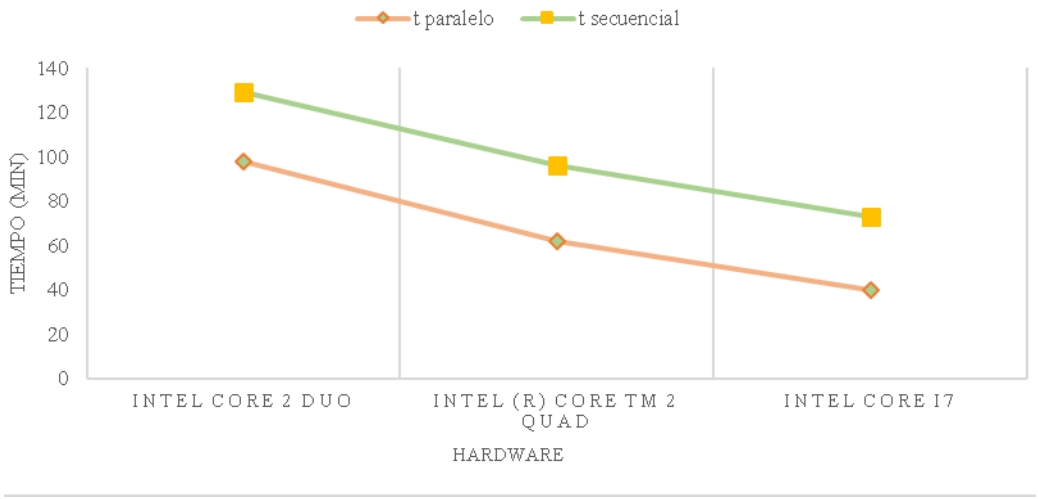

Figura 2: Comportamiento del tiempo de ejecución secuencial y paralelo para varias arquitecturas de hardware

Figura 3: Resultados de la aceleración y la eficiencia tomados para varias arquitecturas de hardware

Para el tercer escenario fueron comparados los tiempos de ejecución de FuzzyPred en su versión paralela, con la cantidad de registros de la base de datos. Este escenario tiene como objetivo conocer cuánto mejora el tiempo de ejecución en función del tamaño de las bases de datos. Los parámetros de configuración se encuentran en la tabla 8 ylos resultados de este escenario son mostrados en la tabla 9.

Tabla 8: Parámetros de configuración del escenario 3.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 2, Abril-Junio, 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 117-133

Parámetros de FuzzyPred	*operador de lógica difusa=zadeh, conectiva=aleatoria, escala de valores=0-12, ejecuciones=50, iteraciones por ejecución=1000, objetivo=valor de verdad, algoritmo metaheurístico=búsqueda aleatoria, base de datos=Quacke.*
Características de la PC	Intel Core i3 -2100 CPU 4 Gb de RAM

Es posible plantear que el tiempo de ejecución de FuzzyPred es mayor para Quacke ya que la base de datos es mucho más grande (contiene más registros), tal y como se muestra en la tabla 9, además, el tamaño de los datos es un factor relevante en el tiempo de ejecución del algoritmo. Es importante señalar que la correspondencia entre tamaño de los datos y tiempo de ejecución es proporcional, ya que disminuye el tiempo de acuerdo al tamaño de la base de datos. Las métricas aceleración y eficiencia son medidas inversamente proporcionales, tal y como se muestra en la figura 4.

Tabla 9: Tiempos de ejecución obtenidos para diferentes bases de datos.

Base de Datos	Tiempo de ejecución secuencial	Tiempo de Ejecución Paralelo	Cantidad de Registros
Quacke	91 min	62 min	2178
Stulong	82 min	54 min	1417
Bolts	75 min	26 min	40

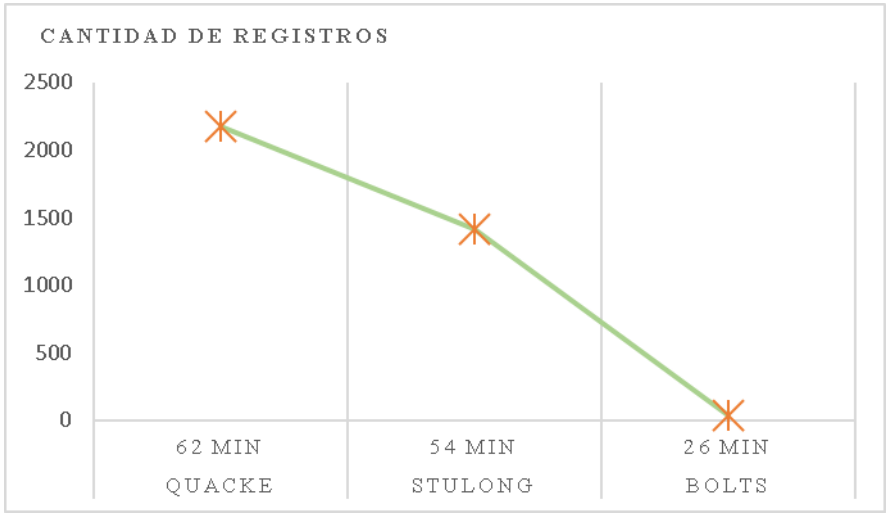

Figura 4: Resultados del tiempo de ejecución paralelo frente a varias bases de datos

Conclusiones

En este trabajo se presenta un diseño de paralelismo de datos implementado en la biblioteca Java Parallel. El diseño paralelo propuesto logra reducir el tiempo de ejecución de la versión secuencial. El modelo se basa en dividir la cantidad de datos en dependencia de la cantidad de procesadores de la arquitectura de hardware. Los resultados experimentales confirmaron que la versión paralela logra reducir en un 10 % a la versión secuencial. En los experimentos se comprueba que los resultados mejoran de acuerdo a las características de hardware, de forma proporcional y que el algoritmo es más rápido en bases de datos más pequeñas. Se recomienda realizar otras pruebas con bases de datos más grandes y en otros tipos de arquitecturas de hardware.

Referencias

ANANTH, G., G. ANSHUL, K. GEORGE and K. VIPIN. Introduction to Parallel Computing. Addison Wesley, 2003.

CHAPMAN B, G. JOST and R Van der Pas. Using OpenMP: Portable Shared Memory Parallel Programming Scientific and Engineering Computation. The MIT Press.Massachusetts Institutte of Technology. ISBN 978-0- 262-53302-7. pp 349. 2008.

BEDDO, V. Applications of Parallel Programming in Statistics. Tesis Doctoral. Universidad de California, Los Angeles, 2002.

CERUTO T. Método para el descubrimiento de predicados difusos en forma normal en bases de datos utilizando metaheurísticas. Tesis Doctoral. CUJAE, La Habana. 2014.

CERUTO T, O. LAPEIRA, A. ROSETE and R. ESPÍN.Discovery of fuzzy predicates in database. Advances in Intelligent Systems Research (AISR Journal), vol. 51, No 1, pp. 45-54, ISSN 1951-6851, Atlantis Press, 2013.

CERUTO T, O. LAPEIRA, A. ROSETE and R. ESPÍN. Knowledge Discovery of fuzzy predicates in database. In Soft Computing for Bussines Intelligence vol 537, pp 187-196, ISSN 1860-949X. Springer Berlin Heidelberg, 2014.

CERUTO T. Método para obtener predicados difusos a partir de datos utilizando metaheurísticas. Tesis de Maestría, CUJAE, La Habana, 2010.

Dean J, S. GHEMAWAT. MapReduce: simplified data processing on large clusters. Commun ACM, vol 51, pp 107-113, 2008.

EL-REWINI H and ABD-EL-BARR. Advance Computer Architecture and Parallel Processing. John Wiley & Sons, Inc. Hoboken, New Jersey. ISBN 0-471-46740-5, pp 287, 2005.

FARBER, R. CUDA Application Design and Development, Elsevier. ISBN 978-0123884329, pp 336, 2011.

FERNANDEZ A, S. DEL RIO, V. LOPEZ, M. J. DEL JESUS and F. HERRERA. Big Data with Colud Computing:an insight on the computing enviroment, Map Reduce and programming frameworks. WIREs Data Mining and Knowledge Discovery.John Wiley and Sons, vol 4, pp 380-409, 2014.

FOSTER, I. Designing and building parallel programs. Addison Wesley Publishing Company. ISBN 978-0201575941,pp 430, 2003.

GUEREQUETA , R. and A. VALLECILLO. Técnicas de Diseño de Algoritmos. Universidad de Malaga. ISBN 84-7496-666-3, 1998.

HAN J, and M. KAMBER. Data Mining: Concepts and Techniques. Morgan Kaufmann. ISBN 978-0123814791, 2011.

HARIRI S, and M. PARASHAR.Tools and Enviroments for Parallel and Distributed Computing. John Wiley & Sons. ISBN 0-471-33288-7, pag 229, 2014.

HERNÁNDEZ J, M.J. RAMÍREZ and C. FERRI. Introducción a la minería de datos. Pearson Educacion, S.A. Madrid, Universidad Politécnica de Valencia, Departamento de Sistemas Informáticos y Computación.ISBN 84-205-4091-9, pp 680, 2004.

HOEGER, H. Introducción a la Computación Paralela. Centro Nacional de Cálculo Científico Universidad de Los Andes, Mérida (Venezuela)–Ce-CalCULA, 2011.

KAMINSKY, A. Parallel Java: A unified API for Shared Memory and cluster Parallel Programming in 100% Java. IEEE International Parallel and Distributed Processing Symposium, Rochester Institute of Technology, Departament of Computer Science, Rochester, New York, EUA. 2007.

KAMINSKY, A. The Parallel Java 2 Library Parallel Programming in 100 % Java. Rochester Institute of Technology, Departament of Computer Science, Rochester, New York, EUA. 2015.

KANTARDZIC, M. Data Mining: concepts, models, methods and algorithms. John Wiley & Sons, ISBN 978-0-470-89045-5, pp529, 2011.

LAPEIRA, O. Rediseño de FuzzyPred 1.0. Tesis de Diploma CUJAE, La Habana, 2012.

MOSTAFA, A.-E.-B. and H. EL-REWINI. Fundamentals of Computer Organization and Architecture. Inc. Hoboken, New Jersey. ISBN 0-471- 46741-3. pp 290. 2005.

PAS, R. An Overview of OpenMP 3.0. In., 2009.IWOMP. Tu Dresden (Alemania). Disponible en http://iwomp.zih.tu-dresden.de/downloads/2.Overwiew_OpenMP.pdf

REINDERS, J. Intel threading building blocks-outfitting C++ for multi-core processor parallelism. OReilly Media. ISBN 978-1449390860, pp 336, 2007.

TOSINI, M. Introducción a las arquitecturas paralelas. 2011. Disponible en http://bit.ly/rH9pylQ

OpenMP.org . The OpenMP API Specification for parallel programming.Disponible en www.openmp.org

VENUGOPAL K, K.G. SRINIVASA and L. M. PATNAIK. Soft Computing for Data Mining Applications. Springer Berlin Heidelberg: Springer-Verlag. ISBN 978-3-642-00192-5, pp 354, 2009.