ANÁLISIS DE ALGORITMOS PARALELOS PARA LA TAREA DE MINERÍA DE DATOS ASOCIACIÓN

ROSA MARÍA ZAMBRANO BURBANO

UNIVERISIDAD DE NARIÑO

FACULTAD DE INGENIERIA

PROGRAMA DE INGENIERIA DE SISTEMAS

SAN JUAN DE PASTO

2014

ANÁLISIS DE ALGORITMOS PARALELOS PARA LA TAREA DE MINERÍA DE DATOS ASOCIACIÓN

ROSA MARÍA ZAMBRANO BURBANO

Trabajo de grado presentado como requisito parcial para optar al título de Ingeniero de Sistemas

Asesor:

M.Sc. MANUEL ERNESTO BOLAÑOS

UNIVERISIDAD DE NARIÑO

FACULTAD DE INGENIERIA

PROGRAMA DE INGENIERIA DE SISTEMAS

SAN JUAN DE PASTO

2014

NOTA DE RESPONSALIBILDAD

“Las ideas y conclusiones aportadas en la tesis de grado son responsabilidad exclusiva de sus autores”

Artículo 1º, de Acuerdo Nº 324 de octubre 11 de 1966, emanado por el Honorable Consejo Directivo de la Universidad de Nariño.

Artículo 13, Acuerdo Nº 005 de 2010 emanado del Honorable Consejo Académico.

Nota de aceptación

_____
_____
_____
_____
_____
_____
_____

_____
Firma del Asesor

_____
Firma del Jurado

_____
Firma del Jurado

San Juan de Pasto, Octubre 2014

AGRADECIMIENTOS

Al finalizar este trabajo de grado mi más profundo y sincero agradecimiento a todas aquellas personas que con su ayuda han colaborado en la realización del presente trabajo, en especial al Ingeniero Manuel Ernesto Bolaños director, por la orientación, el seguimiento y la supervisión continúa de la misma, pero sobre todo por la paciencia y el apoyo recibido a lo largo de este año.

De igual manera agradecer a los Ingenieros Ricardo Timarán y Andrés Calderón, por su visión crítica de muchos aspectos sobre el trabajo y por sus consejos.

Gracias también a mis compañeros, que me apoyaron y me permitieron entrar en su vida durante la carrera y convivir dentro y fuera del salón de clase.

A todos ellos, muchas gracias.

DEDICATORIA

Gracias a Dios, por todas las Bendiciones.

Con todo mi cariño y mi amor para las personas que hicieron todo en la vida para que pudiera lograr mis sueños, por motivarme y darme la mano cuando sentía que el camino se terminaba, a ustedes por siempre mi corazón y mi agradecimiento. Especialmente a mi Mamita Rosa María y mis hermanos los mejores del mundo Graciela, Claudio, Andrés, Gema y Gina Paola.

RESUMEN

La presente investigación describe el análisis de algoritmos paralelos y el estudio de los que estén implementados, mediante pruebas de tiempos de ejecución, donde se obtuvo la implementación de los algoritmos Apriori (Bodon) y Fp-Growth (Mahout).

El análisis de estos algoritmos tiene como finalidad, identificar el de menor tiempo de ejecución, con una cantidad de datos determinada. Estas pruebas se establecieron dentro de un clúster heterogéneo que está formado por varios computadores que se utilizan para el procesamiento de datos.

Finalmente, esta investigación se desarrolló con el propósito de buscar algunas características importantes que se utilicen como referencia para elaborar e implementar un nuevo algoritmo y de esta manera obtener mejores resultados, al momento de ser ejecutados y así utilizar menos herramientas informáticas.

ABSTRACT

This research describes the analysis of parallel algorithms and the study about the implemented ones by testing execution times, where the implementation of the algorithms Apriori (Bodon) and Fp-Growth (Mahout) were obtained.

The analysis of these algorithms is intended to identify the lowest execution time, algorithm using a certain amount of data. These tests were set within a heterogeneous cluster witch consists of several computers that are used for data processing.

Finally, this research was conducted in order to find some important features that are used as a reference to develop and implement a new algorithm and thus get better results at the time of execution. Therefore, use less computer tools.

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE SIGLAS

		Pág.
MISD	Multiple Instruction Single Data (Múltiples Instrucciones Un Solo Dato)	37
MIMD	Multiple Instruction Multiple Data (Múltiples Instrucciones Múltiples Datos)	39
OLTP	Procesamiento de Transacciones en Línea	21
ODS	Base de Datos Operacionales	21
SIMD	Single Instruction Multiple Data (Una Instrucción Un Flujo de Datos)	38
SGBD	Sistema Gestor de Base de Datos	43
SISD	Flujo único de instrucciones flujo único de datos	36
FIM	Frequent Itemset Mining (Conjuntos Frecuentes de Minería)	55

GLOSARIO

Algoritmo: Es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad [1].

Análisis: Es una exploración objetiva y exacta de algún hecho o dato, descomponiéndolo para su estudio o valoración [2].

Base de Datos Distribuidos: Es una colección de datos distribuidos en diferentes nodos de una red de computadoras. La Base de Datos Distribuida representa naturalmente la estructura geográfica descentralizada de una organización, aumentan la disponibilidad de los datos, reducen el tráfico de comunicación y es justificable, además, por el abaratamiento de los costos en el equipamiento y la infraestructura de comunicaciones de las redes de computadoras [3].

Datos: Es una colección de hechos significativos y pertinentes que permiten dar información detallada sobre alguna información recolectada.

Data Warehouse: Es un repositorio de datos de muy fácil acceso, alimentado de numerosas fuentes, transformadas en grupos de información sobre temas específicos de negocios, para permitir nuevas consultas, análisis y decisiones [4].

Granuralidad: Es la ejecución de procesos o aplicaciones a nivel del sistema operativo o a nivel de la red de ordenadores [5].

Grafo: Es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto [6].

Items: Conjuntos de valores que se repiten. Un conjunto de ítems, o itemsets, es aquel cuyos elementos son atributos, variables o campos de cualquier base de datos [7].

Lexicográfica: Es obtener información suficiente pero no total [8].

Minería de Datos: Se define como un proceso que reúne un conjunto de herramientas de diversas ciencias tales como Estadística, Informática, Matemáticas, Ingeniería, entre otras que persigue extraer conocimiento oculto o información no trivial de grandes volúmenes de datos [9].

Overhead: Es la reducción de comunicaciones [10].

Reglas de Asociación: Se utilizan para descubrir hechos que ocurren en común dentro de un determinado volumen de datos [11].

Procesos: Es un programa de bloque secuencial, con flujo de control propio [12].

Transacción: Es un conjunto de operaciones que forman una única unidad lógica de trabajo. Aunque se realicen varias operaciones (actualizaciones, consultas, eliminaciones, etc.) desde el punto de vista del usuario la operación es única [13].

INTRODUCCION

Actualmente, la información que se maneja en el mundo ha obligado a emplear nuevas técnicas de procesamiento basadas en algoritmos complejos, que permiten tener mayor rapidez y eficiencia en el tratamiento de los datos.

Conocer y analizar la información que tiene una empresa es de vital importancia al momento de tomar decisiones que en caso de ser equivocadas implicarían costos adicionales e incluso pérdidas que pueden llevar al fracaso de un negocio. Por tal razón, se está haciendo muy común el uso de herramientas informáticas que permitan manejar grandes volúmenes de información para sustraer datos relevantes. Este proceso se conoce como "Descubrimiento de Conocimiento en Base de Datos – DCBD".

La minería de datos es una de las etapas del proceso DCBD que mediante diferentes tareas y algoritmos permite descubrir patrones ocultos y útiles para la toma de decisiones a partir de grandes volúmenes de datos.

El objetivo de la investigación fue analizar los algoritmos paralelos para la tarea de minería de datos asociación que se ha propuesto hasta el momento con el fin de determinar la forma como paralelizan esta tarea.

Este documento se divide en capítulos. En el siguiente capítulo se describe cual fue el problema a resolver. En el capítulo II, se describe el marco teórico, en el capítulo lll, se muestran algunos antecedentes respecto a la investigación, en el capítulo lV, se desarrolla el proceso de pruebas de los algoritmos implementados en el clúster heterogéneo, en el capítulo V se analizan los resultados finales que se obtuvo durante las pruebas y finalmente, en el capítulo Vl, se muestra las conclusiones del trabajo.

Adicionalmente, es de importancia el anexo que corresponda los resultados que se obtuvo al ser ejecutado el algoritmo Apriori (bodon) y Fp-Growth (mahout) generado durante el proceso de pruebas.

DESCRIPCIÓN DEL PROBLEMA

PLANTEAMIENTO DEL PROBLEMA

Las demandas de cómputo de alto rendimiento en las áreas de investigación científica y tecnológica día a día demandan mayores capacidades de procesamiento. Muchos proyectos académicos se ven estancados debido a que no tienen o no se utilizan los medios y algoritmos de procesamiento computacional adecuado para realizar cálculos complejos, los cuales requieren demasiado tiempo y no se obtiene el resultado deseado, esto implica la frustración del proceso.

En la minería de datos una forma de representar la información es mediante reglas de asociación ya que se utilizan para descubrir hechos que ocurren en común dentro de un determinado conjunto de datos. Este modelo encuentra asociaciones importantes entre variables de una base de datos, mediante el hallazgo de conjuntos de ítems frecuentes que son usados para generar reglas de asociación. El cálculo de estos ítems ha motivado muchas investigaciones debido al alto costo computacional que requiere, para ello se pretende realizar el análisis de los diferentes algoritmos paralelos propuestos para el cálculo de itemsets frecuentes para así poder lograr un mayor desempeño, permitiendo así la división de un problema en subproblemas de forma que se puedan ejecutar de forma simultánea en varios procesadores. Los algoritmos paralelos son importantes puesto que son rápidos en tratar grandes tareas de computación mediante las técnicas secuenciales.

OBJETIVO GENERAL

Analizar los diferentes algoritmos paralelos propuestos para la tarea de minería de datos asociación, con el fin de determinar el más adecuado.

OBJETIVOS ESPECÍFICOS

•Investigar los algoritmos paralelos para la tarea de asociación.

•Analizar y clasificar los algoritmos investigados, teniendo en cuenta las técnicas de paralelización que se utilizan.
•Construir un plan de pruebas para los algoritmos investigados.
•Evaluar el desempeño de los algoritmos.
•Identificar la forma más adecuada de elaborar y presentar la información recolectada.
•Realizar la respectiva documentación del trabajo desarrollado.

JUSTIFICACIÓN

En la actualidad la cantidad de información que puede contener una base de datos es enorme. Esta información puede ser aprovechada por la minería de datos para crear modelos que permitan predecir situaciones o comportamientos útiles apoyando la toma de decisiones en cualquier área de investigación, sin embargo, el costo computacional que implica su desarrollo ha impedido un rápido avance en el tema. Es aquí que con el análisis de los algoritmos paralelos de asociación se puede obtener beneficios, tales como:

- Mejorar el rendimiento en el tiempo del cálculo de los itemsets.
- Obtener resultados en un límite de tiempo aceptable.
- Incrementar el número de los recursos que la tecnología actualmente ofrece.
- Explotar las últimas tecnologías de cómputo.

ALCANCE Y DELIMITACIONES

Este proyecto de grado está enfocado a realizar el análisis de algoritmos paralelos para la minería de datos, iniciando por la apropiación de conocimiento sobre las técnicas de paralelización, integración y algoritmos de complejidad, y realizar el análisis de algoritmos paralelos, propuestos para la tarea de minería de datos, este análisis será tanto teórico como práctico utilizando un conjunto de datos sintético o de un repositorio de la web, con el fin de obtener resultados favorables en cuanto a la eficiencia y tamaño de la memoria de los algoritmos investigados.

METODOLOGÍA

En la realización de este trabajo de investigación se utilizó una metodología que consistió en 4 fases:

Fase 1. Apropiación de las técnicas de paralelización y algoritmos de complejidad.

Fase 2. Investigaciones propuestas de algoritmos de paralelización para la asociación de minería de datos.

Fase 3. Clasificación según las técnicas de paralelización.

Fase 4. Análisis de los algoritmos utilizando técnicas de ingeniería de software.

A partir de las pruebas obtenidas se registraron de manera detallada para obtener la documentación final.

1. MARCO TEÓRICO

1.1 MINERIA DE DATOS

La minería de datos es un conjunto de herramientas y técnicas de análisis de datos que por medio de la identificación de patrones extrae información, novedosa y potencialmente útil de grandes bases de datos que puede ser utilizada como soporte para la toma de decisiones [14].

En la minería de datos en gran parte las técnicas son una combinación directa en tecnología de bases de datos y data warehousing, con técnicas de aprendizaje automático y de estadística.

El propósito del proceso de minería de datos es extraer conocimiento de grandes bases de datos, esto se logra utilizando algoritmos. Los siguientes pasos del proceso de minería de datos, son:

- ✓ Selección de datos
- ✓ Depuración de datos
- ✓ Enriquecimiento de datos
- ✓ Transformación de datos
- ✓ Preparación de un conjunto de casos
- ✓ Construcción del modelo de minería de datos

-Selección de datos

La selección de datos para el proceso de minería de datos se divide en dos partes, como primera instancia es la localización de los datos, y la segunda instancia consiste en la identificación de los datos, a continuación se describe cada instancia.

Localización de los datos: la minería de datos puede ser implementada en casi cualquier base de datos. Las bases de datos recomendadas para la minería de datos son:

- ✓ Data Warehouse: El data warehouse es ideal para la minería de datos, los procesos que ya posee el data warehouse seleccionan, limpian, enriquecen y transforman los datos, estos procesos son muy parecidos a los utilizados por la minería de datos. El data warehouse ha sido diseñado para hacer Queries que manejan un alto volumen de información.

- ✓ Data Mart: El data mart es un subconjunto del data warehouse encapsulado para propósitos específicos del negocio. El data mart solo contienen la información necesaria para satisfacer la investigación. Como los data mart se modelan según las necesidades de los usuarios o de las empresas, la mayor parte de los data mart no son adecuados para la minería de datos. Sin embargo se puede construir un data mart diseñado específicamente para la minería de datos [15].

- ✓ Bases de datos OLTP (procesamiento de transacciones en línea): Son conocidas como bases de datos operacionales, no han sido optimizadas para el tipo de recuperación que se requiere en la minería de datos; Los impactos de ejecución como el acceso y velocidad de transacción se pueden dar en otras aplicaciones que depende de la optimización de actualización de alto volumen de tales bases de datos.

- ✓ Bases de datos operacionales (ODS): Se usan para procesar y consolidar volúmenes grandes de datos típicamente manejados por bases de datos OLTP. Las bases de datos operacionales son típicamente usadas como un buffer de datos entre los datos de OLTP y aplicaciones que requieren acceso a tales datos.

La minería de datos es una búsqueda de datos que se basa en la experiencia, no una búsqueda que pretenda obtener inteligencia de los datos.

Identificación de datos: Para la identificación de los datos se debe tener en cuenta la calidad de los datos escogidos ya que estos determinan finalmente la eficacia de los modelos de la minería de datos. El proceso de identificar los datos para su uso en la minería de datos va en paralelo con el proceso de selección de datos, utilizado en los data warehouse.

-Depuración de datos: La limpieza de datos es el proceso en el que se afirman los propósitos de la minería de datos, los datos son uniformes desde el punto de vista del uso de la llave y los atributos. Identificar y corregir

información perdida, limpiar los registros, son aspectos de la depuración de datos. Para Limpiar los datos debidamente se requiere los pasos siguientes:

- ✓ **Verificación de consistencia de las llaves:** verificar que los datos probados son consistentes a través de todos los datos pertinentes.

- ✓ **Verificación de las relaciones:** verificar que las relaciones entre casos se conforman con reglamentos de negocios puntualizados. Las relaciones que no soportan reglamentos de negocio definidos pueden sesgar los resultados de un modelo de minería de datos.

- ✓ **Uso de atributos y verificación de alcance:** generalmente, la calidad y exactitud de un atributo está en proporción directa a la importancia de los datos. Verificar que los atributos utilizados se están usando tal y cual es la información que se tiene en la base de datos, y que el alcance o campo de los atributos seleccionados tienen el significado al escenario para ser modelado.

- ✓ **Análisis de datos:** verificar que los valores almacenados en los atributos son adecuados según el escenario a evaluar.

-Enriquecimiento de datos: es el proceso de añadir nuevos atributos, tales como campos calculados o datos de fuentes externas, a los datos ya existentes.

El enriquecimiento de datos es importante si está pretendiendo minar datos. Se puede añadir información a tales datos, de las fuentes de industria de partes externas normalizadas para hacer que el proceso de la minería de datos sea más exitoso y confiable, o proporcionar atributos derivados adicionales para una comprensión mejor de relaciones indirectas.

-Transformación de datos: es el proceso de cambiar la forma o estructura de los datos existentes.

Las orientaciones para la transformación de datos son similares en la minería de datos y los data warehouse, y una cantidad grande del material de

referencia existe para la transformación de datos en los entornos de los data warehouse.

Muchos algoritmos de minería de datos funcionan mejor al trabajar con un número pequeño de atributos discretos, tales como rangos de sueldos, antes que atributos continuos, tales como sueldos reales.

Ciertos proveedores de algoritmos de minería de datos pueden transformar los datos, en datos discretos de forma automática, usando una variedad de los algoritmos diseñado para crear rangos discretos basadas en la distribución de datos dentro de un atributo continúo.

-Preparación de un conjunto de casos: el conjunto de casos es usado para construir el conjunto inicial de reglas y patrones que sirven de base para un modelo de minería de datos. Por frecuente se desea tener tantos casos de prueba como sea posible cuando se crea un modelo de minería de datos, asegurándose de que el conjunto de casos represente la densidad y distribución del conjunto de casos de producción, para ello se realiza una selección de casos.

- ✓ **Selección de casos:** al preparar un conjunto de casos, se debe seleccionar que los datos no sean ambiguos, en la medida de lo posible para obtener el resultado que pueda ser modelado. La ambigüedad del conjunto de datos escogido debe ser directamente proporcional al ancho del enfoque según el escenario que se desea representar.

-Construcción del modelo de minería de datos: la construcción consiste en la selección de un algoritmo de minería de datos que se ajuste a las metas que se desean obtener al evaluar el conjunto de casos. Este, a su vez, genera un conjunto de valores que reflejan unas o más vistas estadísticas con el comportamiento del conjunto de casos [13].

1.2 DATOS, INFORMACIÓN Y CONOCIMIENTO

Datos: los datos son en esencia números o texto que puede ser procesado en una computadora, en la actualidad las organizaciones acumulan grandes cantidades de datos en distintos formatos y en distintas bases de datos, entre las que se incluyen datos operacionales o transaccionales en las que se almacenan costos, ventas, inventarios, contabilidad, etc. [16].

Información: los patrones, asociaciones o relaciones entre los datos proporcionan información, por ejemplo el análisis de transacciones de un punto de venta nos pueden dar información sobre qué cantidad de productos se han vendido y durante cuánto tiempo.

Conocimiento: la información puede ser convertida en conocimiento partiendo de patrones históricos [17].

1.3 ALGORITMO

Un algoritmo se define como una secuencia de instrucciones que representan un modelo de solución para un determinado problema. O bien como un conjunto de conocimientos que realizadas en orden conducen a obtener la solución de un problema. El diseño de algoritmos requiere creatividad y conocimientos profundos de la técnica de programación. Los algoritmos deben ser precisos, definidos, finitos y finalmente debe producir un resultado [18].

1.4 PROCESAMIENTO EN PARALELO

Se define como procesamiento paralelo al concepto de agilizar la ejecución de un programa mediante su división en fragmentos que pueden ser ejecutados simultáneamente, cada uno en un procesador. De este modo un programa que se ejecute en *n* procesadores podría ejecutarse *n* veces más rápido que usando un solo procesador. Las aplicaciones con suficiente paralelismo hacen buen uso de múltiples procesadores. El problema radica en identificar las porciones de programa que puedan ser ejecutadas independiente y simultáneamente en procesadores separados; esto en realidad es complejo, ya que se encuentran aplicaciones que podrían ser ejecutadas en paralelo y sin embargo se ralentizan al ser paralelizadas en un sistema particular. Por ejemplo, un programa que tarda 4 segundos en ser ejecutado en una sola máquina podría tardar 1 segundo de procesamiento en cada uno de los cuatro procesadores disponibles en una red, pero no se lograría nada si la coordinación entre dichos procesadores tardase más de 3 segundos. Si se encuentra en alguno de estos casos se observa que el procesamiento paralelo puede proporcionar el rendimiento de un supercomputador, aplicado a algunos programas que realizan complejas operaciones u operan en grandes bloques de datos. Y lo que es más, ello se puede lograr con hardware relativamente barato [19].

1.5 ALGORITMO PARALELOS

Conjunto de tareas que interaccionan mediante el intercambio de mensajes a través de canales.
Es un algoritmo que puede ser ejecutado por partes en el mismo instante de tiempo por varias unidades de procesamiento, para finalmente unir todas las partes y obtener el resultado correcto [19].

1.5.1 Factores que presentan los algoritmos paralelos

-Tiempo de ejecución: el tiempo de ejecución es el índice de prestaciones más intuitivo. Es un parámetro absoluto pues permite medir la rapidez del algoritmo sin compararlo con otro [20].

-Eficiencia: es el grado de aprovechamiento de los procesadores para la resolución del problema.

- Escalabilidad: es la capacidad de un determinado algoritmo de mantener sus prestaciones cuando aumenta el número de procesadores y el tamaño del problema en la misma proporción. En definitiva la escalabilidad suele indicarnos la capacidad del algoritmo de utilizar de forma efectiva un incremento en los recursos computacionales [38].

1.5.2 Diseño de algoritmos paralelos

-Diseño de Algoritmos Paralelos

El diseño se divide en cuatro etapas, las cuales son:

- ✓ Partición: esta etapa busca subdividir el problema lo más que se pueda, con el fin de buscar oportunidades de paralelismo, una buena partición subdivide tanto los cómputos (cálculos) como los datos. Existen dos formas de proceder con la descomposición.

- Descomposición del dominio: su área de trabajo se centra en los datos, se determinan las particiones apropiadas de los datos para después darle seguimiento a los cálculos asociados a los mismos.

- Descomposición funcional: realiza las divisiones a nivel de cálculos y luego se encarga de los datos.

A la hora de particionar hay que tener en cuenta lo siguiente:

- Hay que evitar que los cálculos y almacenamiento sean redundantes, de lo contrario el algoritmo no se podría aplicar a problemas grandes.
- Tratar en la medida de lo posible que las tareas sean de tamaño equivalentes ya que esto facilita el balanceo de la carga de los procesadores.

- El número de tareas debe ser proporcional al tamaño del problema, esto permite que el algoritmo sea escalable.

✓ Comunicación: se define la comunicación de un algoritmo en paralelo en dos fases, primero se delimitan los canales que conectan las tareas que requieren datos con las que lo poseen. Y en el segundo se especifica la información o mensajes que deben ser enviado y recibidos en estos canales.

El algoritmo se implementa según el tipo de máquina, entre los mecanismos de sincronización son utilizados los semáforos, los cuales son usados para controlar el acceso a la memoria compartida y coordinar las tareas.

En esta etapa se considera lo siguiente:

- Todas las tareas deben efectuar el mismo número de operaciones de comunicación, de no suceder esto pueda ser que el algoritmo no sea extensible a problemas mayores ya que habrá cuellos de botella.
- La comunicación entre tareas debe ser tan pequeña como sea posible.
- Las operaciones de comunicación deben poder proceder concurrentemente.
- Los cómputos de diferentes tareas deben poder proceder concurrentemente.

✓ Agrupación: en esta fase va de lo abstracto a lo concreto y se revisa el algoritmo tratando de producir un algoritmo que se ejecute eficientemente sobre cierta clase de computadoras. Mediante la agrupación de tareas se puede reducir la cantidad de datos a enviar y así reducir el número de mensajes y el costo de comunicación.

Es importante tener en cuenta algunas características, tales como:

- Verificar si la agrupación redujo los costos de comunicación.
- Verificar que las tareas resultantes tengan costos de cómputo y comunicación similares.
- Analizar si es posible reducir aún más el número de tareas sin introducir desbalances de cargas o reducir la extensibilidad.

- ✓ Asignación: después de haber establecido las etapas anteriores, se procede a la última etapa, que consiste en determinar en qué procesador se ejecutará cada tarea. La asignación de tareas pueden ser dinámicas o estáticas.

En la asignación estática las tareas son asignadas a un procesador al comienzo de la ejecución del algoritmo en paralelo y corren ahí hasta el final.

En la asignación dinámica se hacen cambios en la distribución de las tareas entre los procesadores a tiempo de ejecución, esto quiere decir que hay migración de tareas a tiempo de ejecución, esto ayuda a balancear la carga del sistema y reducir el tiempo de ejecución Sin embargo, el costo de balanceo puede ser significativo y por ende incrementar el tiempo de ejecución.

Entre los sistemas de balanceo se encuentran los siguientes:

- Balanceo Centralizado: Un nodo ejecuta y mantiene el estado global del sistema.
- Balanceo Completamente Distribuido: cada procesador mantiene su propia visión del sistema intercambiando información con sus vecinos y así hacer cambios locales.
- Balanceo Semi-Distribuido: divide los procesadores en regiones, cada una con un algoritmo centralizado local.

A continuación, se observa en la figura 1, las 4 etapas del diseño de los algoritmos en paralelo [20].

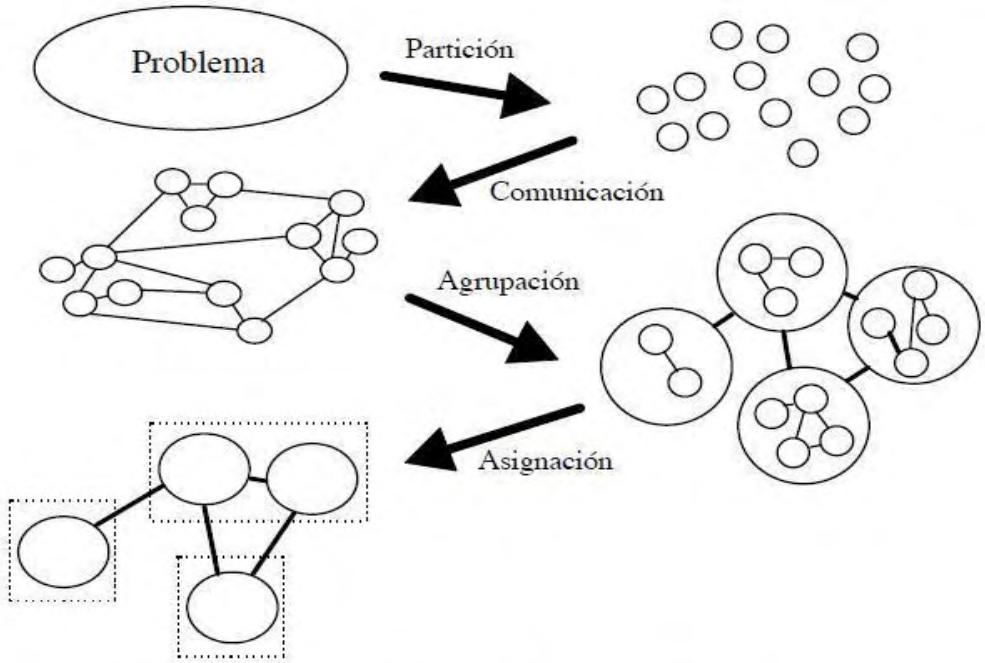

Figura 1. Etapas de diseño del algoritmo [20]

1.5.3 Evaluación de algoritmos paralelos: Para la evaluación de algoritmos paralelos se debe tener en cuenta el objetivo principal de la evaluación, las estrategias usadas en dicha evaluación, las formas de mérito con el fin de obtener una buena evaluación. Para ello se describe a continuación cada ítems, así:

-Objetivo de la evaluación. Se debe tener en cuenta cuál es el algoritmo más rápido, entre varios algoritmos de investigación, también se debe tener en cuenta la eficacia con la que se está utilizando la máquina, finalmente se observa cómo se comporta el algoritmo al variar alguna de las características de la máquina o del problema.

-Estrategias de evaluación. Para esta aplicación se tiene en cuenta cuatro estrategias de evaluación las cuales son:

- ✓ Evaluación experimental: esta etapa requiere de la disponibilidad de hardware y la implementación del algoritmo, no permite especular sobre el comportamiento del algoritmo ante modificaciones de la máquina.

- ✓ Evaluación mediante simulación: esta estrategia solicita una especificación detallada de muchos detalles de la máquina y del algoritmo.

- ✓ Evaluación mediante modelo analítico: Esta fase muestra la construcción del modelo. Esta etapa en más eficiente y rápido y también facilita la comparación de algoritmos.

-Formas de mérito. Esta evaluación de estrategia presenta diferentes formas de méritos tales como:

- ✓ Tiempo de ejecución:

Es la figura de interés para el usuario, no permite evaluar la eficacia del algoritmo.

- ✓ Speed Up: es el tiempo de ejecución en un procesador sobre un tiempo de ejecución en *p* procesadores.

$$S = \frac{T1}{Tp} \qquad (1)$$

Para el tiempo *T1* existen dos formas para calcularlo la primera es de forma *Absoluto* que se calcula usando el mejor algoritmo secuencial y de segunda es de forma *Relativo*, se calcula ejecutando el algoritmo paralelo en un solo procesador.
El speed up resuelve la eficacia con que se está utilizando la máquina.

Otra forma de calcular el Speed Up es:

$$T1 = Tseq + Tpar \quad (2)$$

Tseq es el tiempo empleado en cálculos no paralelizables.
Tpar es el tiempo empleado en cálculos perfectamente paralelizables entre cualquier número de procesadores [12].

Con esta aplicación se obtiene la siguiente ecuación:

$$Tp = Tseq + \frac{Tpar}{P} + O(p) \quad (3)$$

O(p) es el overhead debido a:
Desequilibrio de carga y sobrecarga de comunicación y/o sincronización.

Estas formas de mérito definen la escalabilidad como un sistema (arquitectura + algoritmo) es escalable si es capaz de usar de forma satisfactoria un número creciente de procesadores.

Existen muchas propuestas de métodos para determinar si un sistema es escalable o no. Cada método toma como punto de partida una definición más precisa de uso satisfactorio de un número creciente de procesadores [21].

-Megaflops (o Megaflop/segundo)

Evalúa la potencia de cálculo del sistema. Se obtiene como el cociente entre el coste teórico aritmético del algoritmo (medido en flops, es decir, operaciones en coma flotante) y el coste temporal del algoritmo (en segundos) [22].

- Eficiencia. Mide el grado de utilización de los procesadores y se calcula como el cociente de:

$$E_p^n = S_p^n / p \quad (4)$$

- Escalabilidad. Degradación de la eficiencia del algoritmo al aumentar el número de procesadores y el tamaño del problema. Se dice que un algoritmo es escalable para un problema de tamaño (local) *k* cuando $E_p^n = constante$, *n*= *kp*, p=1,2,…La escalabilidad de los algoritmos, si se da, suele alcanzar para valores de *k* elevados [23].

1.5.4 Técnicas de paralelización automáticas: se describen algunas técnicas orientadas a extraer o facilitar la extracción del paralelismo de bucles tanto para sistemas multiprocesadores como para máquinas vectoriales. Todas ellas se basan en el grafo de dependencias obtenido previamente por el compilador.

Existen 2 grupos de técnicas: el primer grupo de técnicas son las que están independientes de la arquitectura y están orientadas a facilitar la extracción de concurrencia, entre ellas, están:

-Distribución de bucles: pretende separar partes del bucle que pueden ser vectorizadas o paralelizadas de otras partes que en principio deben ejecutarse de forma secuencial. La técnica de distribución de bucles también se puede aplicar para la vectorización de bucles anidados, para ello existen métodos que consisten en explorar cada uno de los bucles existentes. Una de las posibilidades es empezar por analizar el bucle más interno y se termina por el más externo.

La técnica de distribución de bucles también permite la generación de bucles paralelos *Doall* (ciclos) para aquellas sentencias que quedan incluidas en *n*-blocks acíclicos [24].

-Intercambio de bucles: para el caso de los bucles anidados, se debe cambiar el orden en que los bucles son ejecutados con la finalidad de aumentar la eficiencia del proceso de paralelización.

Esta técnica se utiliza también para encontrar bucles vectorizables dentro de un conjunto de bucles anidados, para este caso se interesa vectorizar el bucle más interno.

-Eliminación de Independencias: aquí se describen algunas técnicas que permiten eliminar y modificar determinadas relaciones de dependencia de bucle. Las técnicas que se describen son las siguientes:

- ✓ Cambio de nombre y expansión escalar: Estas técnicas permiten eliminar dependencias de datos debidas a la reutilización de variables escalares de bucle, esta reutilización provoca antidependencias y dependencias de salida evitables.
  La técnica de cambio de nombre asocia un nombre distinto en cada sentencia que define la variable escalar reutilizada, modificándolo también en todas aquellas sentencias que usan la definición asociada.
  La técnica de expansión escalar, convierte en vector aquellas variables escalares a las que se asigna un nuevo valor en cada iteración de bucle.

- ✓ Refinamiento de nodos: Es utilizada para eliminar dependencias de salida en recurrencias haciendo copias temporales intermedias de las variables que las ocasionan.

El segundo grupo de técnicas están orientadas a la generación de código paralelo para sistemas multiprocesador y en general a resolver problemas de la existencia de recurrencias en bucles secuenciales.

-DOPIPE: Es una técnica que no pretende paralelizar π-blocks cíclicos sino solapar al máximo su ejecución secuencial con la ejecución del resto de π-blocks, reduciendo así el número de procesadores necesarios para ello.

DOPIPE necesita introducir sincronización con el fin de asegurar el correcto solape en la ejecución de π-blocks.

La efectividad de esta técnica depende del algoritmo utilizado para determinar las sentencias que constituyen cada segmento, intentando solapar al máximo la ejecución del segmento determinante del tiempo de ejecución del bucle con el resto de los segmentos. Cada segmento se ejecuta secuencialmente en un procesador y puede ser necesario añadir sincronizaciones a fin de asegurar las relaciones de dependencias entre segmentos.

-DOÁCROSS: Propone que cada iteración de bucle espera a que todos los datos que necesita para ser ejecutada estén libres de dependencias, también asigna iteraciones a procesadores y el inicio de cada una de ellas se retarda

unidades de tiempo respecto a la anterior. DOÁCROSS determina que una iteración no se inicie hasta que todas las anteriores lo hayan sido también.

-Particionado parcial: es un método que permite la paralelización de recurrencias, asignando iteraciones del bucle a procesadores distintos de manera que no existan dependencias entre ellas, el número de tareas viene determinado por el máximo común divisor de las distancias del grafo distintas de cero.

La restricción que mejor representa este método es que todas las dependencias de datos deben de tener asociada una distancia estrictamente mayor que la unidad a fin de obtener paralelismo de recurrencia.
-Contracción de Ciclos: Es un método que permite la paralelización de recurrencias. Se basa en ejecutar en paralelo todas aquellas iteraciones del bucle consecutivas que no dependen de ninguna iteración previa [25].

La paralelización del algoritmo disminuye notablemente el tiempo de ejecución del algoritmo, a mayor grado de paralelización son necesarias mayores cantidades de iteraciones para que el algoritmo converja [26].

1.5.5 Técnicas de paralelización. Las técnicas de paralelización se describen a continuación.

-**Divide y vencerás** (Divide and Conquer). Es una técnica importante y que más se ha aplicado en el diseño de algoritmos eficientes, esta técnica consiste en dividir el problema en subproblemas más pequeños del mismo tipo, con el fin de resolver estos subproblemas de forma separada y combinar los resultados parciales para obtener la solución parcial y finalmente obtener la solución total. Este método es utilizado recursivamente para facilitar el desarrollo [27][28].

A continuación, en la figura 2, se muestra el esquema de la técnica divide y vencerás

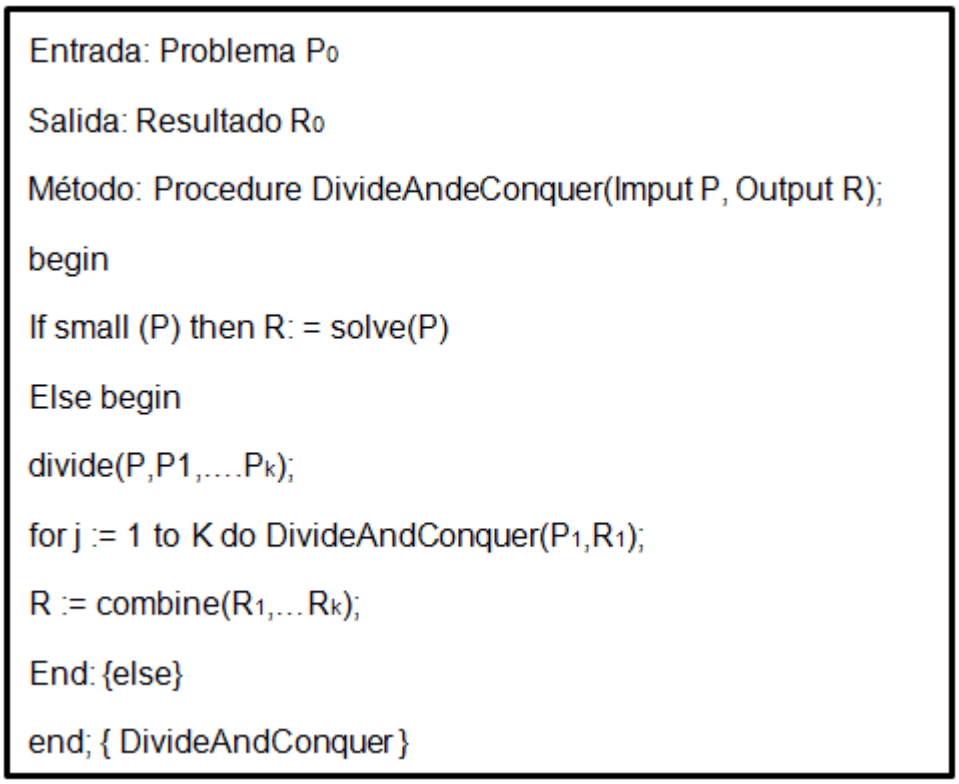

Figura 2. Técnica divide y vencerás [27]

El problema de la clasificación, se puede formular de la siguiente manera definiendo cada uno de los items que se utilizaron anteriormente en el esquema.

Entrada: es una secuencia de n elementos tales como a1, a2,… an y una relación de orden total ≤.

Salida: Es una permutación π de los n elementos a π(1), a π(2),… ,a π(n) tal que a π(i) ≤ a π(i+1) 1 ≤ i ≤ n.

Operación de división (divide): se basan en la división del problema de entrada en cuantos subproblemas de menor tamaño, independientes entre sí y del mismo tipo que el de partida.

Operación combinación (combine): una vez resueltos los subproblemas se hace necesario la utilización de una nueva función que recibiendo como entrada los subproblemas resueltos devuelva el problema original resuelto.

Con esta operación se debe aprovechar la optimiladidad de las soluciones *R1* de los subproblemas *P1* en general.

Función de parada (small): esta función consiste en que deberá discernir sobre la conveniencia de la división de un problema, en caso negativo debe encontrar una solución al problema por otro tipo de método solve que no necesita descomposición ya que este puede estar compuesto por un solo elemento.

-Ramificación y acotación (Branch and Bound): se parte de la creación de un algoritmo de tipo enumerativo, esto es, el problema original se descompone en una serie de problemas parciales de menor tamaño y de mismo tipo que el de partida. Esta técnica trata de maximizar o minimizar el valor de una función sobre una región de búsqueda arbitraria.

Este procedimiento de ramificación y acotación consiste en la repetición sucesiva de generación y comprobación de problemas parciales con el fin de encontrar una solución *I* problema de partida examinando la menor parte posible del árbol de búsqueda [18][20].

-Programación dinámica (Dynamic Programming): permite crear relaciones de recurrencia entre los problemas y sus descendientes, debido al principio de optimalidad ya que la decisión que se toma sólo depende de información local y nunca provoca la generación de resultados erróneos, por lo tanto, basta con generar una decisión en cada estado [29].

A continuación, en la figura 3, se indica el esquema de la técnica de programación dinámica.

```
Entrada: Problema P0
Salida: Resultado f[x1]…[xn]
Método: Procedure DynamicProgramming;
        Var i1, i,2,….. in;
        Begin
          Initialize(f);
            For i, in set (x1) do
              For i2 in set (x2)do
              …….
                 For in in set (xn)do
              F[i1]…[in] i=optimize g(f);
                          0≤i1≤i1
        End; {DynamicProgramming}
```

Figura 3. Técnica de programación dinámica

1.6 ARQUITECTURA PARALELAS

[26] Clasifica los ordenadores en cuatro categorías atendiendo a la multiplicidad de los flujos de instrucciones y datos, sobre estas arquitecturas se pueden establecer, a un nivel superior, muchos modelos de programación. Los modelos básicos que se pueden encontrar, de más actualidad, son el de paralelismo de datos, el de paso de mensajes y el de memoria compartida.

-Arquitectura SISD (Single Instruction Single Data). Este tipo de arquitecturas corresponde a la idea única instrucción ejecutándose sobre un único conjunto de datos, las instrucciones son ejecutadas una detrás de otra. En la figura 4, se muestra el esquema simplificado de la arquitectura SISD [26][30].

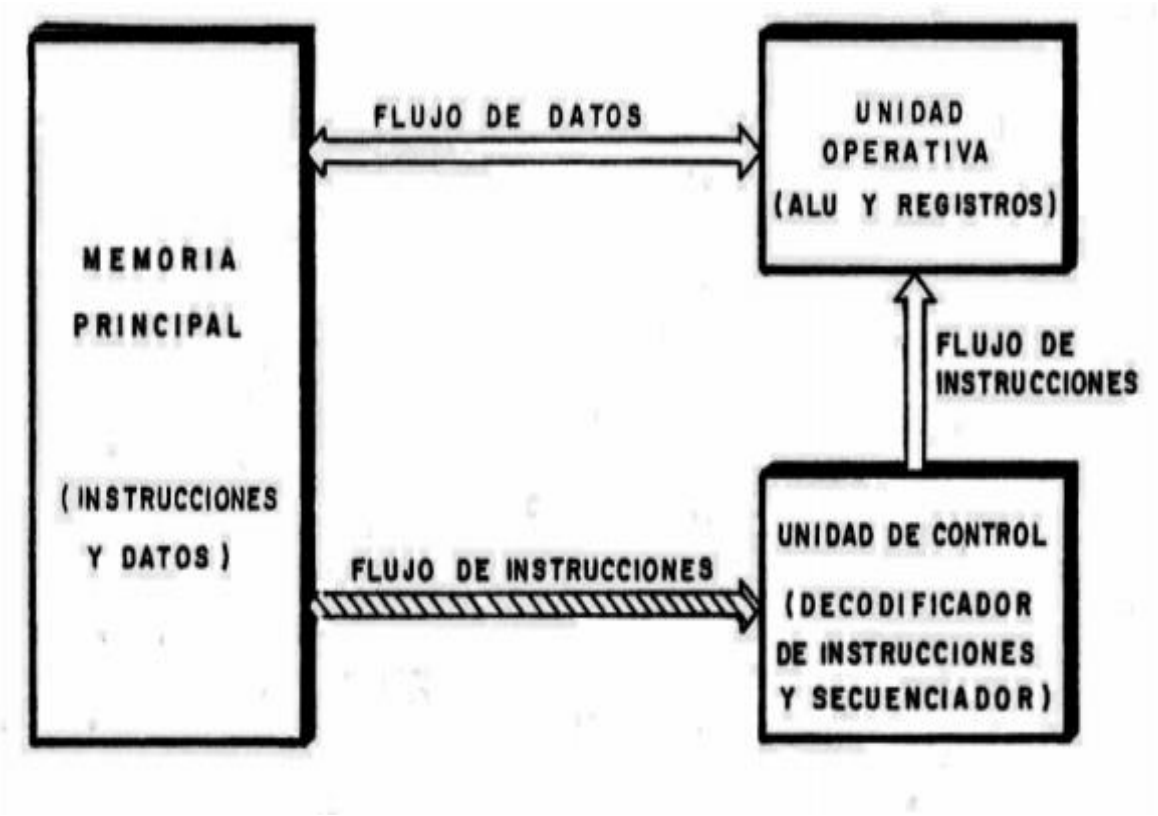

Figura 4. Esquema simplificado de la arquitectura SISD [26]

-Arquitectura MISD (Multiple Istruction Sinlge Data). Un flujo múltiple de instrucciones opera sobre un flujo simple de datos. Cada vez que un dato es extraído de la memoria se procesa múltiple veces, esta arquitectura se conoce como array sistólicos para ejecución conducida de algoritmos específicos. La figura 5 indica la arquitectura MISD de múltiple flujo de instrucciones y flujo único de datos [26][30].

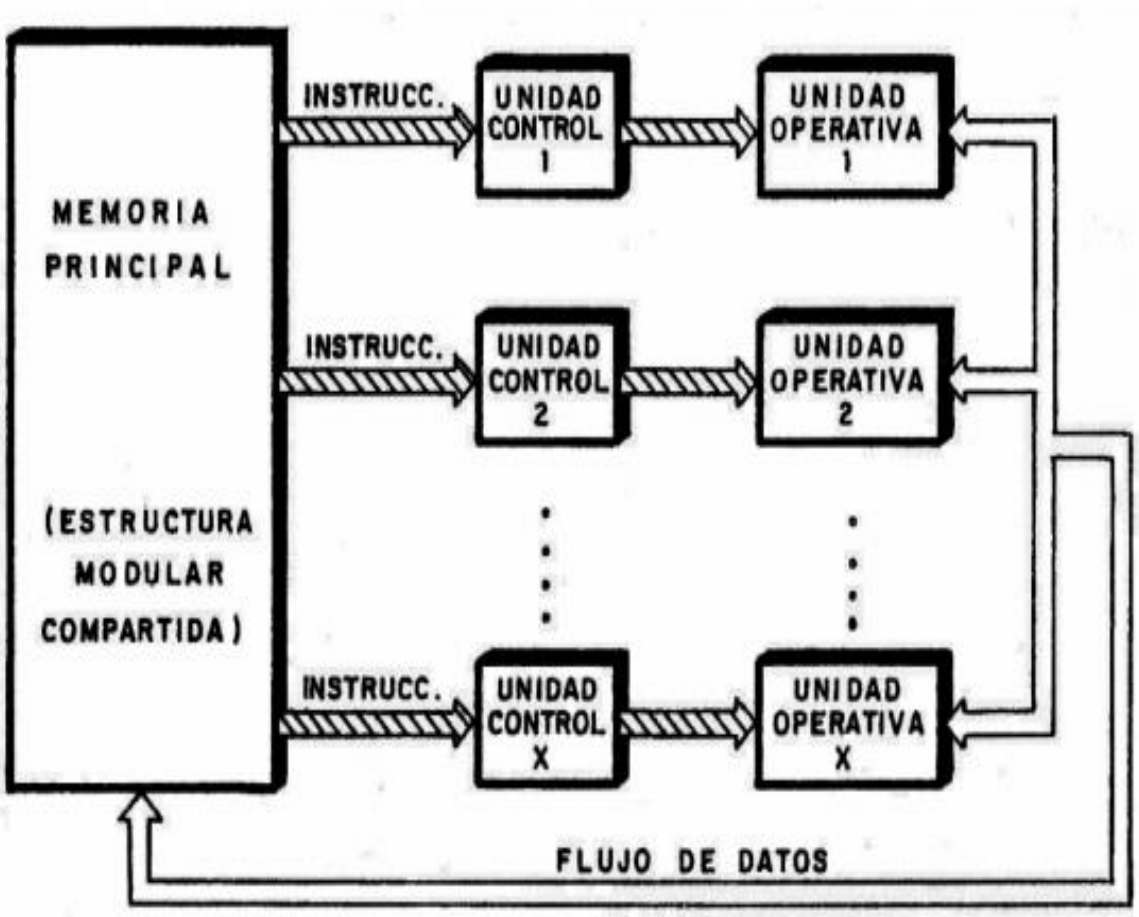

Figura 5. Arquitectura MISD de múltiple flujo de instrucciones y flujo único de datos [26]

-Arquitectura SIMD (Single Instruction Multiple Data). Los computadores que poseen esta arquitectura suelen estar formados por varios procesadores pero con una única unidad de control. Todos los procesadores ejecutan la misma instrucción, bajo las órdenes de la unidad de control, pero cada uno de ellos sobre sus propios datos. Todos los procesadores ejecutan las mismas instrucciones simultáneamente, pero sobre diferentes partes del conjunto de datos a tratar. En la figura 6 se da a conocer la arquitectura SIMD, de flujo único de instrucciones y flujos múltiples de datos [26][30].

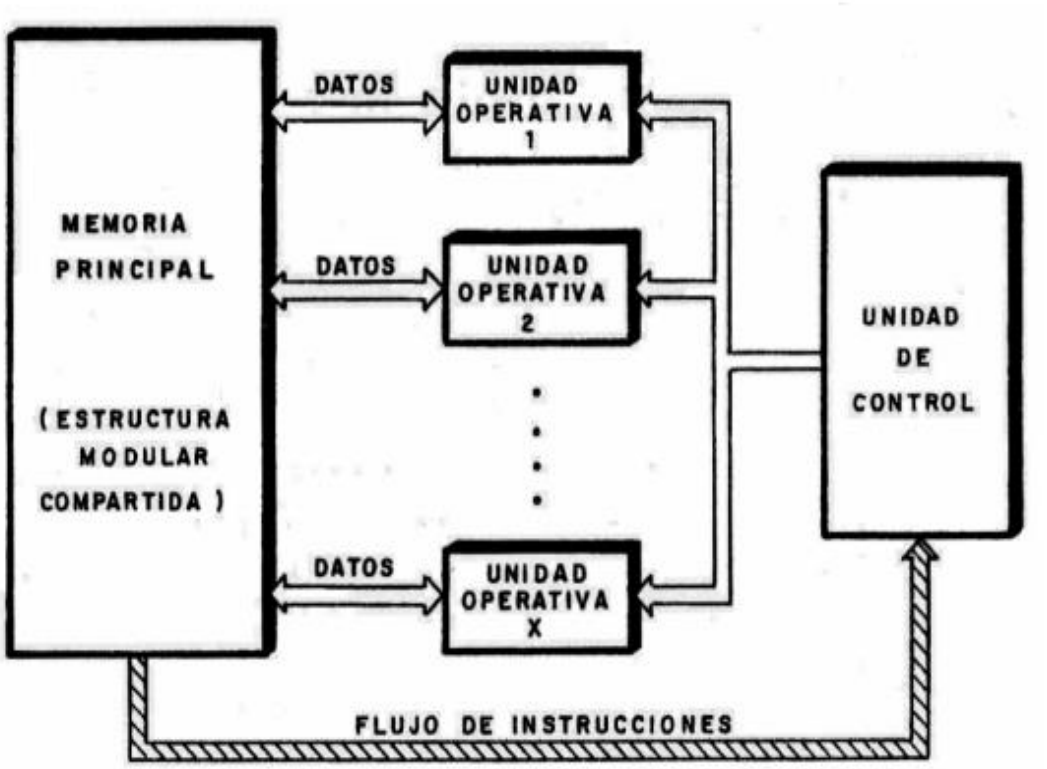

Figura 6. Arquitectura SIMD, de flujo único de instrucciones y flujos múltiples de datos [26]

-**Arquitectura MIMD** (Multiple Instruction Multiple Data). Distintas instrucciones se ejecutan sobre distintos conjuntos de datos, es un conjunto de procesadores/computadores independientes, ejecutando cada uno de ellos su propio flujo de instrucciones y conectados a través de una red de comunicación.

Los sistemas MIMD pueden clasificarse, en base al tipo de conexión y organización de la memoria del sistema, en memoria compartida y memoria distribuida. La Figura 7, muestra la arquitectura de flujo múltiple de instrucciones y flujo múltiple de datos [26][30].

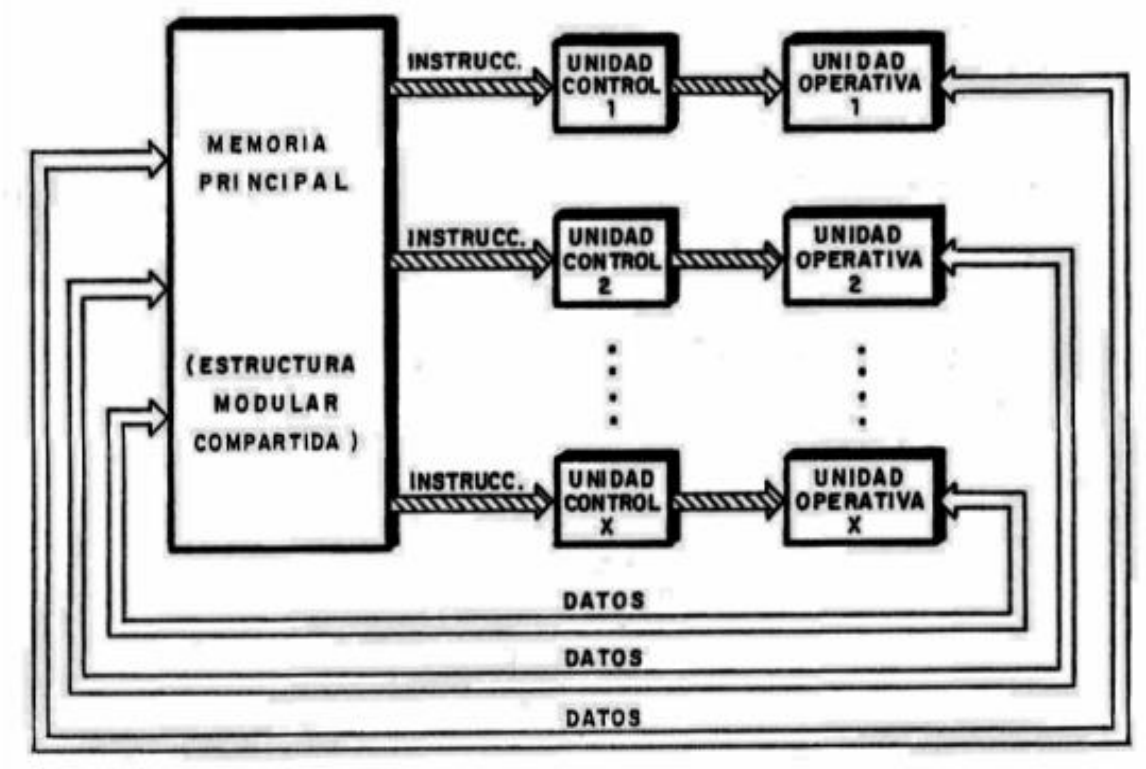

Figura 7. Arquitectura de flujo múltiple de instrucciones y flujo múltiple de datos [26]

1.7 TIPOS DE ALGORITMO DE ASOCIACIÓN

Algunos de los algoritmos que hacen parte de esta investigación son los siguientes:

1.7.1 Algoritmo a priori. El algoritmo Apriori emplea la confianza que es la que mide con qué frecuencia aparece Y en las transacciones que incluyen X y soporte es la fracción de las transacciones que contiene tanto a X como a Y, para encontrar todas las reglas que superen los pasos especificados por el usuario. Sin embargo, con este método se pueden obtener muchas reglas, por lo que un sistema de descubrimiento de conocimiento debe evaluar el interés de éstas y mostrar al usuario un número razonable de reglas interesantes que le sean útiles. Además, se requiere que el usuario sea en cierto grado experto para que encuentre el balance adecuado entre el soporte y la confianza que le devuelva reglas interesantes. La mayoría de los algoritmos de minería de datos requieren establecer muchos parámetros de entrada, los cuales si no se establecen correctamente puede dar lugar a falsos patrones o que sobreestime la importancia de los patrones encontrados [31].

La Minería de datos utiliza algoritmos como Apriori, para explorar los repositorios de datos y extraer conocimiento a partir de los datos.

“El algoritmo Apriori permite encontrar ideas de patrones frecuentes. Este algoritmo utiliza tres pasos para encontrar los conjuntos de elementos frecuentes. Relaciona transacciones originales con Itemsets frecuentes, y finalmente encuentra modelos secuenciales [32]”.

El algoritmo Apriori es usado para descubrir reglas de asociación. La idea básica del algoritmo Apriori es generar un conjunto de datos candidatos de un tamaño particular; y examina la base de datos para encontrar estos elementos y ver si hay grandes cantidades de elementos frecuentes.

La generación de las reglas de asociación a partir de los itemsets frecuentes es casi inmediata. Por este motivo la mayor parte de los algoritmos de extracción de reglas de asociación han centrado su diseño en la búsqueda eficientemente de todos los itemsets frecuentes que se encuentran en una base de datos.

Funcionalidad de Apriori. El algoritmo Apriori realiza varios recorridos

secuenciales sobre la base de datos para conseguir los conjuntos de itemsets relevantes. Apriori genera los itemsets candidatos única y exclusivamente a partir del conjunto de itemsets frecuentes encontrados con la iteración que se presenta a continuación, el cual utiliza el último conjunto *L[k]* de *k-itemsets* relevantes para generar un conjunto de *(k +1)-itemsets* potencialmente relevantes el conjunto de itemsets candidatos *C [k + 1]*). Tras obtener el soporte de estos candidatos, se escoge sólo los que son frecuentes, que se incluirán en el conjunto *L [k + 1]*, este proceso se repite hasta que no se encuentra más itemsets principales [33].

El algoritmo Apriori genera todos los conjuntos que cumplan con la condición de tener un soporte menor o igual al soporte mínimo que se haya establecido previamente.

Flujo de actividades del algoritmo A priori. El flujo de datos se realiza en el momento en que se seleccionan los datos desde un origen, en este caso es una base de datos, como se indica en la figura 8.

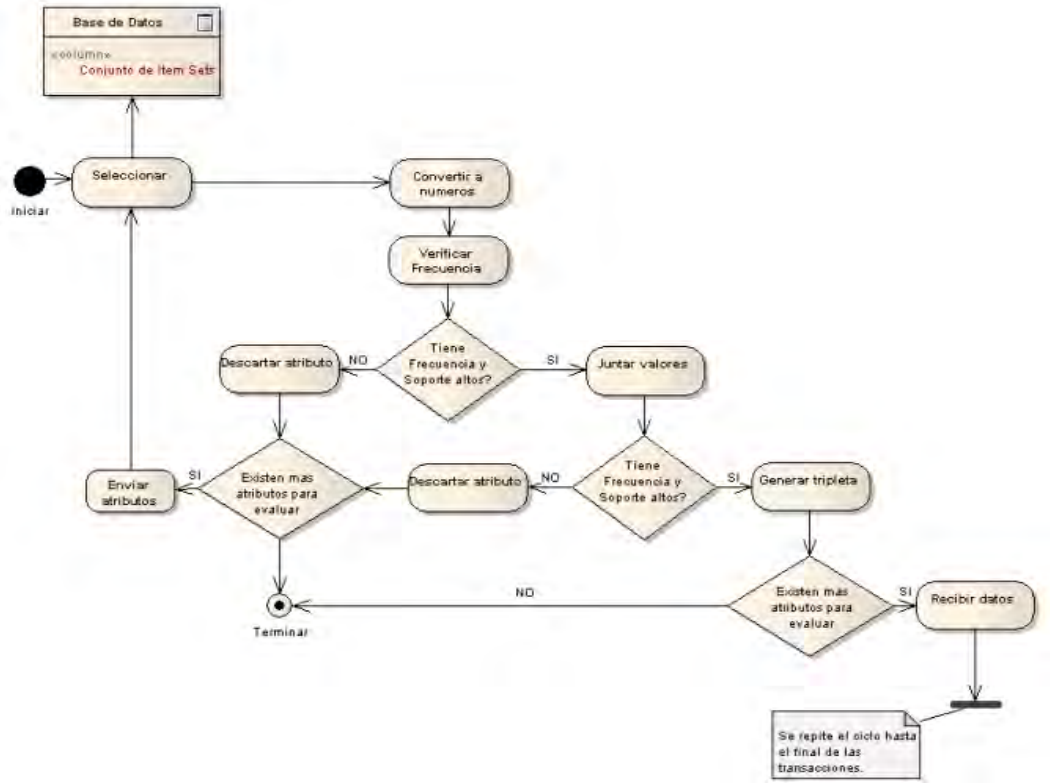

Figura 8. Flujo de actividades de algoritmo a priori [32]

Ventajas del algoritmo A priori. Eficiencia para grandes volúmenes de datos muy elevada. Se presentan pocos casos, en los cuales los datos de entrada, y los resultados intermedios consumen gran cantidad de recursos (memoria) [32].

-Ciertos sistemas gestores de bases de datos, SGBD son capaces de ejecutar este algoritmo dentro del núcleo del gestor [32].

1.7.2 Algoritmo CD (Count Distribution). Este algoritmo utiliza el principio simple de permitir "cálculos redundantes en paralelo para disminuir la comunicación". La primera iteración es especial para todas las demás iteraciones. El algoritmo trabaja como sigue.

1. Cada procesador *pi* genera todo *CK*, usando todos los itemsets frecuentes LK-1 creados al final de la iteración k-1.
2. El procesador *Pi* hace una iteración sobre su partición Di y cuenta el soporte local para los candidatos en *CK*.
3. El procesador *Pi* intercambia los conteos locales de *CK* con todos los demás procesadores para crear los conteos globales de *CK*. Los procesadores están forzados a sincronizar.
4. Cada procesador *Pi* calcula *LK* a partir de *CK*.
5. Cada procesador *Pi* decide independientemente si terminar o pasar a la siguiente iteración. Esta decisión será idéntica en todos los procesadores ya que estos tienen idéntico *LK*. En La primera iteración, cada procesador *Pi* genera dinámicamente sus candidatos locales *C1i*, en dependencia de los items que se encuentren presentes en la partición local *Di*. Por consiguiente los candidatos contados por diferentes procesadores pueden no ser idénticos y debe tenerse mucho cuidado en el intercambio de los conteos locales para determinar el *C1* global [41].

1.7.3 Algoritmo DD (Data Distribution). Este algoritmo se describe de la siguiente forma:

Iteración 1: El mismo procedimiento que en CD.
Iteración K>1

El procesador *Pi* genera *CK* a partir de LK-1. Asumiendo que la cantidad de procesadores es *N*, cada procesador se queda con la *n*-enésima parte de los itemsets formando el subconjunto candidato C_K^i y esto puede ser llevado a cabo sin comunicación alguna entre los procesadores, en este caso los itemsets son asignados usando un round-robin. Los conjuntos C_K^i son disjuntos y su unión es el *Ck* original.

2. El procesador P^j desarrolla el conteo de soporte para los itemsets en su conjunto de candidatos local C_K^I usando datos locales y datos enviados por otros procesadores.
3. Al final de la iteración, cada procesador P^j calcula L_K^I usando el C_K^I local. Nuevamente, los L_K^I son disjuntos y su unión es L_k.
4. Los procesadores intercambian L_K^I para que cada procesador tenga el L_k completo para la generación del C_{k+1} de la siguiente iteración. Este paso requiere que los procesadores se sincronicen. Habiendo obtenido el L_k cada procesador puede decidir independiente e idénticamente si terminar o continuar a la próxima iteración [41].

1.7.4 Algoritmo CD (Candidate Distribution). El algoritmo de distribución de candidatos particiona los datos y los candidatos de tal forma que cada procesador puede proceder independientemente. En algún paso *l*, donde *l* es determinado heurísticamente, este algoritmo divide los itemsets frecuentes del conjunto L_{l-1} entre los procesadores de tal forma que un procesador P^j puede generar un único $\boldsymbol{C_m^i}$ ($m \geq l$) independiente de los otros procesadores ($\boldsymbol{C_m^i} \cap \boldsymbol{C_m^j} = \varphi$, $i \neq j$).

Al mismo tiempo, los datos son particionados tal que cada procesador puede contar los candidatos en C_m^i independiente de los demás procesadores.

1.7.5 Algoritmo FP-GROWTH (Frequent Pattern Growth). Se basa en el crecimiento de ítems o patrones frecuentes utilizando una estructura compacta y eficiente de datos denominada FP-Tree [31]. En un primer recorrido de la colección de datos, se obtienen los conjuntos frecuentes de ítems de tamaño *1.* En un segundo recorrido de la colección de datos, se inserta cada transacción, con los ítems ordenados descendentemente de acuerdo a su soporte, en una estructura de datos compacta denominada *FP-tree*, también conocida como árbol de prefijos [34]. Este algoritmo encuentra los itemsets frecuentes sin generar candidatos, para cada ítem se determina su *conditional pattern base.* Este patrón condicional de un ítem *p* cualquiera está formado por los caminos en el *FP-Tree* que incluyen a los prefijos del nodo que almacenan al ítem frecuente *p*. Posteriormente se crea un árbol *FP* con los elementos de cada base que cumplan con el *soporte* mínimo especificado [35].

El algoritmo FP-Growth es el proceso de obtención de todos los patrones frecuentes de un conjunto de transacciones, mediante la generación del *árbol FP* que comprime las transacciones y el recorrido recursivo de éste.

-**Creación del Árbol FP Inicial.** La generación del *árbol FP* inicial es probablemente la parte más complicada del algoritmo. Ya que la altura depende de la cardinalidad del conjunto de ítems únicos mayor entre todas las transacciones, esta se puede dividir en dos fases:

-Procesamiento inicial de datos.

En esta fase, se transforma el conjunto de transacciones desde el formato usado en la mayoría de las bases de datos transaccionales, esto es, pares ordenados (transacción, ítem), a un formato útil para el proceso de llenado de datos, donde cada transacción es un arreglo ordenado de ítems, y los datos se guardan en un arreglo ordenado de transacciones (normalmente de manera lexicográfica).

Los pasos del procesamiento son los siguientes:

-Calcular soportes de los ítems: se calcula la cantidad de veces que aparece cada ítem en el conjunto.

-Eliminar ítems con soporte bajo: se eliminan las entradas de los ítems que no cumplen con el soporte mínimo (si un ítem no cumple con él, no puede ser parte de un patrón que lo cumpla). Esto se hace con el fin de lograr un llenado más rápido del árbol.

-Aplanar transacciones: se transforman los pares ordenados (transacción, ítem) en arreglos de ítems. El identificador de la transacción se descarta.
-Ordenar transacciones: se ordenan las transacciones de manera lexicográfica.
A continuación la figura 9 muestra el llenado de Árbol *FP.*

```
public void generarNodos(FPNode padre, int profundidad, Rango rango, int[][] datos){
        while(hayDatos(rango)){
                (nuevo_rango, item)= buscar(profundidad, rango, datos);
                        nNodo=crearNodo(padre,item,rango.size());
                        generarNodos(nNodo,profundidad+1,nuevo_rango,datos);
                }
        }
        public FPNode crearNodo(FPNode padre, String item, int conteo){
                nodo=new FPNode(padre, item, conteo);
                lista[item].ultimo().sgte=nodo;
}
```

Figura 9. Llenado de árbol FP [35]

Es importante saber que generarNodos y crearNodo son métodos del *Árbol FP*, es decir, la información que se genere en ellos será agregada inmediatamente a la instancia del árbol. El método crearNodo envuelve el proceso de generación de Nodos FP y genera un nuevo nodo que es añadido a la lista enlazada correspondiente.

El método generarNodos particiona los datos a cada nivel de profundidad y genera el nodo correspondiente.

En la figura 10, se enseña un ejemplo de un patrón base condicional y un FP-Tree condicional para el ítem "*m*" dado en el FP-tree [36].

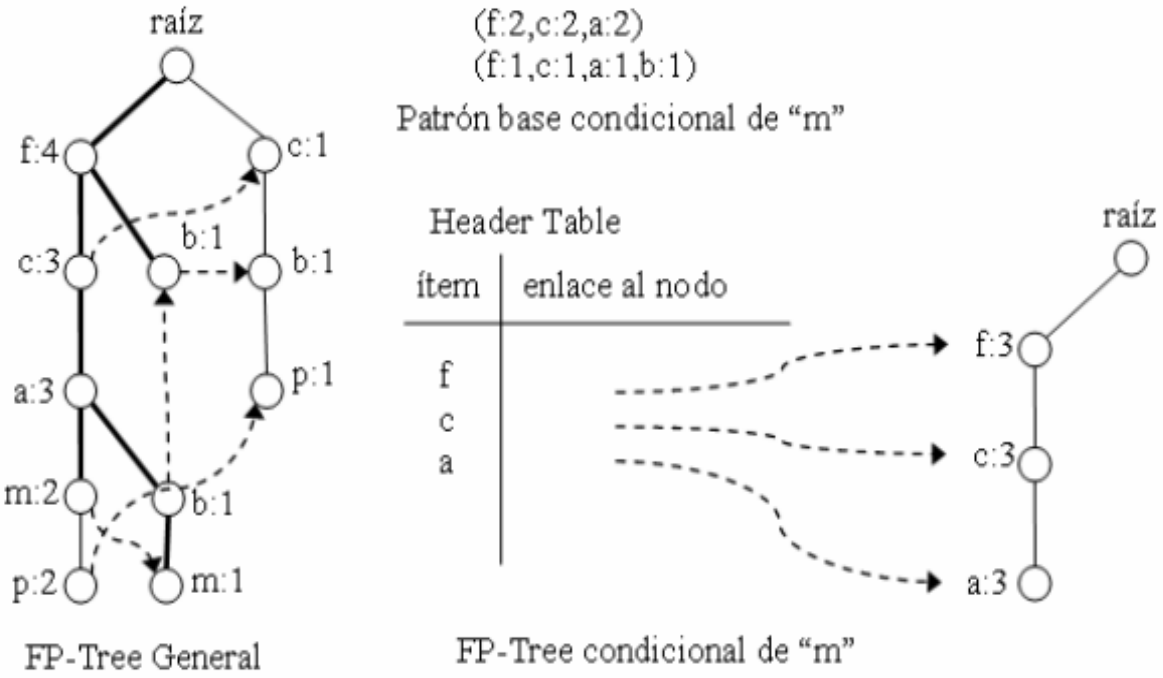

Figura 10. Ejemplo de patrón base condicional [36]

- Recorrido Recursivo: **La figura 11, muestra el algoritmo de podado del árbol**

```
public void buscarPatrones(String patron,int soporte_minimo, FPTree arbol, Lista patrones){
        for(<Cada item contenido en el arbol>){
                if(item.soporte()>soporte_minimo){
                        patrones.add(patron+" "item+" "+item.soporte());
                        BuscarPatrones(patron+" "+item,soporte_minimo,
                        arbol.podar(item),patrones);
                }
        }
}
```

Figura 11. Recorrido recursivo [36]

A continuación, la figura12, indica un ejemplo de minado recursivo de FP-Growth [36].

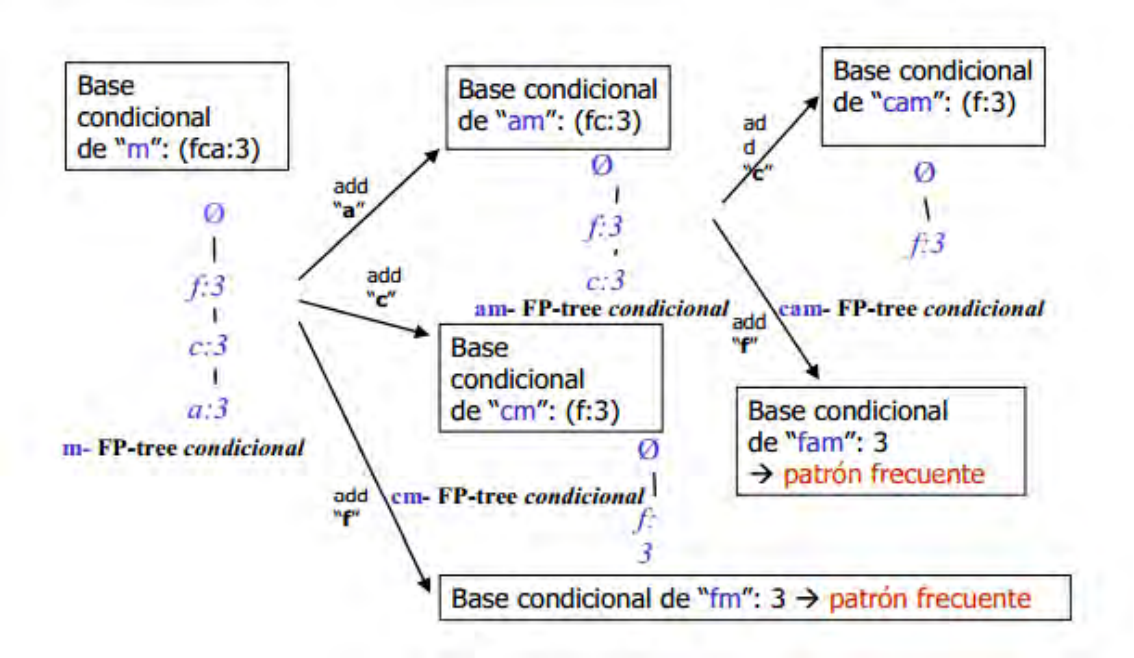

Figura 12. Recursivo minado FP-Growth [36]

En la primera llamada al método, la variable patrón es un String vacío y la variable patrones es una lista vacía. El método agregará a la lista cada patrón+ítem que cumpla con el soporte mínimo y se llamará a sí mismo, con el ítem agregado al patrón inicial, además de entregar un árbol podado según el ítem correspondiente. El podado recursivo permite encontrar los patrones frecuentes en que participa un ítem y sus antecesores. Por el hecho de aplicar este algoritmo sobre todos los items del árbol se obtendrá los patrones frecuentes de éste [36].

1.7.6 Algoritmo equipasso. Es un Algoritmo que se basa en los operadores de algebra relacional Associator y EquiKeep EquiKeep, aseguran [38] para el cálculo de conjuntos de ítems frecuentes. Optimiza las técnicas de aplicación para mejorar el rendimiento.

Descripción del algoritmo equipAsso. En el primer paso del algoritmo se cuenta el número de ocurrencias de cada ítem para determinar los 1-conjuntos de ítems frecuentes *L1*conjunto de tamaños*.* En el subsiguiente paso, se aplica el operador EquiKeep para extraer de todas las transacciones en *D*, los conjuntos de ítems frecuentes tamaño 1, haciendo nulos el resto de valores. Luego, a la relación resultante *R*, se aplica el operador Associator para generar todos los conjuntos de items tamaño *2 (Is = 2)* hasta máximo tamaño *n*, donde *n* es el grado de *R*. Finalmente, se calculan todos los conjuntos de ítems frecuentes *L*, contando el soporte de las diferentes combinaciones generadas por Associator en la relación *R*'. La figura 13 muestra el algoritmo Equipasso [39].

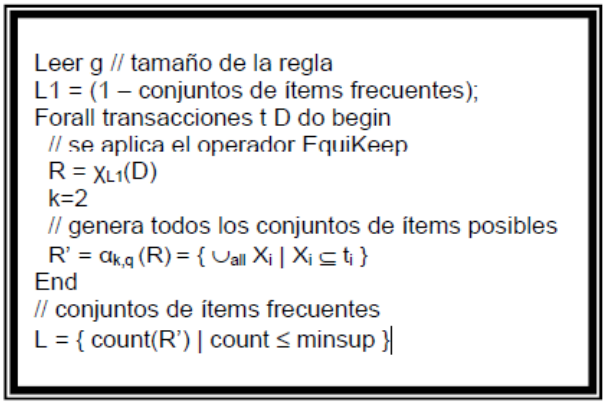

Figura 13. Algoritmo equipAsso[39]

Algoritmo SQL-EquipAsso. A continuación, se muestra el algoritmo que utiliza las nuevas primitivas SQL para minar reglas de Asociación, el algoritmo EquipAsso de la figura 13, se implementa con la cláusula SELECT de la siguiente manera [39].

```
SELECT <ListaAtributosTablaDatos>, count(*) AS soporte INTO R'
FROM D
EQUIKEEP ON L1
ASSOCIATOR RANGE k UNTIL g
GROUP BY <ListaAtributosTablaDatos> HAVING count(*) >= minsup
```

1.7.7 Algoritmo de Ais (Agrawal Imielinski & Swami). Es el primer algoritmo que se desarrolló con base a obtener reglas de asociación, la representación del conocimiento es útil para caracterizar las regularidades que se pueden encontrar en grandes bases de datos.

En AIS, una regla de asociación es una implicación de la forma *X=>Y* \s,c\ donde *X* es un itemsets, *Y* es un ítem no incluido en *X*, *s* representa la relevancia de la regla y *c* su fiabilidad.

El soporte y la confianza de las reglas de asociación se presentan como fracciones. La confianza de la regla de asociación *X=>Y* es la proporción de transacciones con *X* que contienen también a *Y* mientras que el soporte de la regla es la proporción de transacciones de la base de datos que contienen $X_u Y$.

Los itemsets candidatos se generan y su soporte obtiene conforme se recorre la base de datos utilizando un contador. Hay que destacar que los itemsets se almacenan ordenados lexicográficamente [40].

La figura 14, indica el algoritmo AIS

L[k]es el conjunto de itemsets relevantes que contienen k items.

C[K] es el conjunto de k-itemsets (itemsets de k elementos) candidatos, aquéllos que pueden llegar a ser relevantes si tienen la relevancia mínima.
L[1] = {large 1-itemsets}
k = 2
Mientras L[k-1] ≠ 0
C[k] = 0;
Para cada transacción t ε D
Lt = Conjunto de (k-1)-itemsets relevantes contenidos en t
Para cada itemset lt ε Lt
Ct = Extensiones de lt contenidas en t (*)
Para cada candidato c ε Ct
Si c ε C[k]
Incrementar el contador asociado a c en C[k]
Si no
Añadir c a C[k] (contador=1)
L[k] = { c ε C[k] | contador(c)≥ MinSupport }
k++
Solución: $_u$ L[k]

Figura 14. Algoritmo AIS [40]

A partir de los conjuntos relevantes de items encontrados en una transacción, se generan los candidatos difundiendo éstos con items que se encuentren en la transacción.

1.7.8 Algoritmo cba (Classification Based on Associations). Fue el primer algoritmo en el que se formalizó de manera efectiva la integración de conceptos de clasificación y asociación en una única implementación. El algoritmo suele ser utilizado como referencia para el desarrollo de otros trabajos más recientes relativos a la implementación de algoritmos de clasificación basada en asociación, donde se realizan estudios de casos para evaluar la eficiencia de dichos algoritmos.

El algoritmo posee dos componentes principales: un generador de reglas y un constructor de clasificadores. Por lo tanto, el CBA puede ser caracterizado como un algoritmo de dos etapas, pues la obtención de las reglas y del clasificador se realiza en momentos distintos.

El generador de reglas de CBA, llamado CBA-RG, consiste en una adaptación del algoritmo Apriori, para así poder encontrar reglas de clasificación, donde se realizan múltiples recorridos por el conjunto de datos de instrucción para hallar los conjuntos de ítems frecuentes, el procedimiento utilizado para obtener los ítems frecuentes es prácticamente el mismo empleado en Apriori, pero estos son denominados conjuntos de reglas de items frecuentes y poseen la siguiente notación <(condset, condsupCount), (y, rulesupCount)>, donde condeset es un conjunto de items, condsupCount es el soporte de dicho conjunto y una etiqueta de clase y rulesupCount es el soporte de la regla de clasificación, el soporte de la regla se refiere al número de registros en los que los ítems de condset ocurren junto con la clase indicada en y. De esta forma, una regla de ítems frecuentes puede representar una regla de clasificación como se muestra a continuación:

condset -> y

Durante el proceso de generación de las reglas, el algoritmo proporciona únicamente una regla para representar un conjunto de reglas de items que poseen el mismo condset, La regla elegida va a ser aquella que presente el mayor valor de la medida de confianza. En el caso de un escenario en el que más de una regla con el valor máximo de confianza, el algoritmo elige una

regla de forma aleatoria para representar el conjunto de reglas de ítems que poseen el mismo condset [37].

1.8 HERRAMIENTAS SECUENCIALES

Una vez consultados y estudiados los anteriores algoritmos, se toma la iniciativa de buscar algunas de las implementaciones secuenciales. Entre ellos se escogió el algoritmo Apriori y Fp-Growth.

Los algoritmos que se describen a continuación son utilizados en la metodología que se desarrolla más adelante, con el fin de lograr resultados tales como el tiempo de ejecución y el tamaño de memoria con un determinado conjunto de datos.

Apache Mahout es la librería que colecciona diferentes implementaciones entre ellas esta FP-Growth y FMI es la librería de bodon del algoritmo Apriori.

A continuación se describe cada librería.

-Fp-Growth (MAHOUT). Mahout fue construido especialmente como una biblioteca de aprendizaje de máquinas escalable, que implementa muchos enfoques diferentes para el aprendizaje automático.

Actualmente, esta librería contiene implementaciones de algoritmos de clasificación, agrupación, programación genética y filtrado colaborativo que tiene soporte para almacenar su modelo en una base de datos, todos habilitados para el escalado a fin de aprovechar la potencia de la implementación Map Reduce Hadoop (Hadoop es un framework de software que soporta aplicaciones distribuidas bajo una licencia libre. Permite a las aplicaciones trabajar con miles de nodos y datos). Admite resolver problemas como clustering y clasificacion de terabytes de datos sobre miles de ordenadores [43].

Mahout implementa tres áreas que se utilizan en ambientes reales tales como:

-Filtrado Colaborativo: es una técnica popularizada por el Amazon y otros, que utiliza la información del usuario, como el rating, clics y compras para ofrecer recomendaciones a otros usuarios del sitio. En general, el filtrado colaborativo es el proceso de filtrado de información o modelos, que usa técnicas que implican la colaboración entre múltiples agentes, fuentes de datos, las

aplicaciones suelen incluir conjuntos de datos muy grandes. Los métodos de filtrado colaborativo se han aplicado a muchos tipos de datos, incluyendo la detección y control de datos tales como instituciones de servicios financieros que integran diversas fuentes financieras, o en formato de comercio electrónico y aplicaciones web 2.0 donde el foco está en los datos del usuario.

-Clustering: calcula la similitud entre los elementos de la colección, pero su única tarea es agrupar elementos similares. En muchas implementaciones de la agrupación, los elementos de la colección están representados como vectores en un espacio *n*-dimensional.

-Categorización: es etiquetar los documentos no visibles, lo que contribuiría a juntos. Muchos enfoques para la clasificación en el aprendizaje de la máquina que calcula una serie de datos estadísticos que asocian las características de un documento a la etiqueta específica, así creando una plantilla que se puede utilizar posteriormente para clasificar los documentos que no se muestran.

-Apriori (BODON FMI)

Trie es una estructura de datos muy popular en la minería de patrones frecuentes (FIM) algoritmos. Se define como la memoria-eficiente, y permite una rápida construcción y recuperación de información. Muchos *Tries* utilizan técnicas que se pueden aplicar fácilmente en los algoritmos de FIM para así poder mejorar la eficiencia de cada uno de ellos. Ferenc Bodon propone nuevas técnicas para gestión rápida, y especialmente aquellos que pueden ser empleadas en Apriori [44][48].

La implementación de Bodon está basado en *Trie* para la minería de patrones secuencias de elementos frecuentes en una base de datos transaccional.

Trie trata de examinar la estructura de datos, la aplicación y características algorítmicas centrándose principalmente en los datos.

Este algoritmo describe un análisis claro de las características modernas de transformadores, con el fin de entender con facilidad los resultados obtenidos.

El algoritmo Apriori es uno de los algoritmos más antiguos y versátiles de la minería de datos. Con las estructuras que posee este algoritmo demuestra ser

un algoritmo competitivo en el concurso de implementaciones de mineras de conjunto de elementos frecuentes.

Apriori también supera algoritmos basados FP-Growth en bases de datos que incluyen muchos elementos frecuentes, porque la generación de los PF-tree condicionales lleva demasiado tiempo en el proceso de datos transaccionales.

2. ANTECEDENTES

2.1 VERIFICACIÓN FORMAL DE UN ALGORITMO DE PROCESAMIENTO EN PARALELO DE IMÁGENES MÉDICAS

Introducción

En este artículo se muestra la importancia del uso de un método formal conocido como model-checking o probador de modelos, para verificar el buen funcionamiento de un algoritmo paralelo.

Por lo que la técnica de model-checking puede ser usada únicamente cuando los programas son de tamaño finito. El model-checker que se usó en este trabajo es SPIN, el cual es una herramienta que usa el lenguaje de programación Promela (lenguaje similar al lenguaje C).

SPIN verificará que el lenguaje del autómata del programa está incluido en el lenguaje del autómata de la propiedad a verificar, si es así la propiedad se cumple, en caso contrario, SPIN mostrará la secuencia de líneas de código que llevan al error.

En el trabajo se mostró el uso de la herramienta SPIN para garantizar el buen funcionamiento del procesamiento paralelo de un volumen de imágenes médicas. En particular las imágenes se obtienen de un estudio de resonancia magnética. El procesamiento aplicado a cada imagen se refiere a su segmentación para distinguir diferentes regiones del cerebro.

Resultados obtenidos: el procesamiento simultáneo de varias imágenes puede hacerse mediante el diseño de diferentes algoritmos paralelos. El algoritmo más simple es el que reparte equitativamente el volumen de imágenes entre un conjunto de procesadores.

En este caso cada procesador podría determinar, en base a su identificador, el rango de imágenes a procesar, Sin embargo, debido a que los tiempos de segmentación de cada imagen son diferentes, algunos procesadores pueden quedar sobrecargados mientras que otros ya terminaron su tarea.

De esta manera, las imágenes se van asignando a los procesadores conforme estos lo soliciten, balanceando la carga del sistema. Este funcionamiento es también conocido como el protocolo Maestro-Esclavos. Existe un solo procesador que es identificado como el maestro y N procesadores que son los esclavos. Cuando un esclavo no tiene trabajo, solicita una imagen al maestro para realizar su segmentación. Así, las imágenes son distribuidas en base a la capacidad de cada procesador.

Conclusiones

En este trabajo se dio una introducción a la técnica de model-checking para la verificación de un algoritmo de procesamiento paralelo de imágenes cerebrales. Se presentó la teoría básica que soporta esta técnica de verificación y los operadores de la lógica temporal para expresar las propiedades que se desean verificar de un algoritmo.

Mediante la implementación de un algoritmo paralelo con política Maestro-Esclavos para la segmentación de las imágenes de un estudio de resonancia magnética, se ilustraron los pasos para aplicar la verificación. Estos pasos son: especificación del programa en el lenguaje Promela reconocido por la herramienta de model checking SPIN, la especificación de la(s) propiedad(es) a verificar y la verificación de la(s) propiedad(es) usando SPIN, en el algoritmo

Maestro-Esclavos las propiedades verificadas están relacionadas con la terminación correcta del algoritmo, el no interbloqueo entre maestro y esclavos (propiedad verificada por default por SPIN) y el procesamiento completo del volumen de imágenes [45].

2.2 UN ALGORITMO GENÉTICO PARALELO PARA EL PROBLEMA DE STEINER GENERALIZADO

Introducción

Uno de los principales objetivos del trabajo de maestría consistió en investigar la aplicabilidad de los algoritmos evolutivos y en particular de los algoritmos genéticos para resolver problemas de diseño redes de comunicaciones confiables. Tomando en cuenta la dimensión de los complejos problemas de optimización involucrados, se propuso diseñar modelos paralelos de algoritmos genéticos, capaces de manejar la complejidad intrínseca del problema considerado y de utilizar de modo eficiente poblaciones numerosas, en caso de ser necesario.

Luego del estudio teórico de los algoritmos evolutivos y de las estrategias de procesamiento paralelo aplicables para mejorar su eficiencia, y después de identificar y analizar el problema de Steiner Generalizado como un importante problema de optimización subyacente a los problemas de diseño de redes de comunicaciones confiables, este capítulo presenta la primera propuesta

elaborada de un algoritmo genético paralelo específico para la resolución del Problema de Steiner Generalizado.

Resultados obtenidos

Esta sección presenta los resultados obtenidos en los experimentos de validación, configuración y evaluación del algoritmo genético paralelo propuesto sobre un conjunto de problemas de prueba diseñado específicamente con ese fin.

-Instancias de prueba para el Problema de Steiner Generalizado:

Como consecuencia, no se han desarrollado conjuntos estandarizados de instancias de prueba para evaluar comparativamente los resultados de diferentes algoritmos o métodos de resolución del problema. En general, los investigadores que han abordado el problema generalizado han trabajado con instancias de prueba derivadas de los casos más simples, como el caso del problema del árbol de Steiner, o han generado aleatoriamente sus propios casos de prueba.

En el contexto de este trabajo no fue posible recolectar información sobre configuraciones de redes de comunicaciones existentes ni relevar características de diseños que puedan tener aplicación práctica en escenarios reales. Igualmente, la tarea de extender los conjuntos existentes de instancias de prueba para las versiones simples de la clase de problemas de Steiner no es sencilla, requiriéndose un estudio teórico de las propiedades del problema y las características de sus soluciones. Para las pruebas de configuración y validación del algoritmo genético diseñado en este trabajo se utilizó el conjunto de grafos presentados por Robledo (2001). Sin embargo, la reducida complejidad de estos casos de prueba no imponía un reto al algoritmo genético, por lo cual se contempló diseñar un conjunto de problemas más complejos para evaluar los resultados y el desempeño del algoritmo genético paralelo diseñado.

Tomando en cuenta las anteriores observaciones, se optó por diseñar un conjunto de instancias de prueba aleatorias, compuesto de tres problemas, para la evaluación de calidad de resultados y desempeño computacional del algoritmo genético paralelo diseñado. El estudio teórico del diseño de un conjunto de problemas de prueba se propone como una línea inmediata de trabajo futuro.

Los grafos del conjunto de prueba creado se designan por un nombre que referencia a la cantidad total de nodos y el número de nodos terminales. Por ejemplo, grafo 50-15 designa al grafo más pequeño del conjunto de problemas de prueba, que tiene 50 nodos de los cuales 15 son terminales, y así sucesivamente para el resto de las instancias. Las tres instancias diseñadas para evaluar el algoritmo genético fueron generadas seleccionando aleatoriamente topologías de grafos, considerado los nodos como puntos geográficos en un plano euclídeo. Los costos asociados a los enlaces fueron considerados proporcionales a las distancias euclideanas entre nodos para las instancias grafo 75-25 y grafo 50-15, y fueron seleccionados aleatoriamente entre los valores enteros en el intervalo [1, …, 20] para la instancia grafo 100-10. Los requerimientos de conexión para cada par de nodos terminales fueron seleccionados aleatoriamente de modo uniforme entre los valores enteros en el intervalo [0, …, 4] para las tres instancias consideradas.

Conclusiones

-Los resultados obtenidos en la primera propuesta de resolución del Problema de Steiner Generalizado mediante un algoritmo genético paralelo ofrecen perspectivas promisorias sobre la aplicabilidad de las técnicas de computación evolutiva para resolver problemas de optimización vinculados al diseño de redes de comunicaciones confiables.

-Respecto a la performance del modelo de poblaciones distribuidas, evaluada mediante el tiempo de ejecución del algoritmo, el análisis mostró que es posible obtener significativas mejoras de eficiencia respecto al modelo, cuando se utiliza un número reducido de subpoblaciones.

-Estos resultados alientan a continuar la investigación en el área, estudiando tópicos de diseño no considerados en el estudio preliminar, ampliando el horizonte de aplicabilidad y analizando aquellos factores para los cuales no fue posible extraer afirmaciones concluyentes.

-Algunos aspectos dejaron líneas de investigación abiertas para abordar en el futuro inmediato. Tomando en cuenta los promisorios resultados obtenidos, una línea lógica de trabajo futuro se orienta hacia el estudio de otros modelos de

algoritmos evolutivos paralelos para abordar instancias de grandes dimensiones y su comparación con posibles resultados de algoritmos exactos y obtenidos aplicando otras heurísticas de optimización.

-La escasa disponibilidad de recursos computacionales limitó los estudios iniciales de configuración y eficiencia del modelo. Contemplando alternativas para aumentar el número de equipos del clúster será posible llevar a cabo las investigaciones en torno a los aspectos cuya influencia no pudo ser aclarada [46].

2.3 ALGORITMOS PARALELOS PARA DETECTAR SUCESOS DE NOTICIAS EN LÍNEA

En aplicaciones donde existe un flujo ininterrumpido de documentos se requiere mecanismos automáticos que operando a la misma velocidad que el flujo, organicen y filtren la información para su posterior estudio por parte de los usuarios.

Una de estas aplicaciones consiste en la detección y el seguimiento automático de sucesos en flujos de noticias digitales, también conocida como TDT (Topic Detection and Tracking). El problema consiste en determinar si un documento entrante informa sobre un nuevo suceso o forma parte de otros sucesos recogidos por el sistema. La tarea de detección es una abstracción experimental del agrupamiento de noticias. El objetivo del sistema es la agrupar las noticias que abordan un mismo suceso. En esta aplicación hay que tener en cuenta que el conjunto de noticias cambia en el tiempo, pues es necesario modificar el agrupamiento a medida que se van publicando nuevas noticias para mantenerlo actualizado. Por tanto, se requieren algoritmos que sean capaces de realizar la actualización a medida que se publican las nuevas noticias, sin necesidad de empezar nuevamente desde el principio. Estos algoritmos reciben el nombre de algoritmos incrementales o en línea.

Con el creciente de la cantidad de noticias disponibles en línea, una sola computadora no puede resolver el problema de la búsqueda de los sucesos de una parte significativa de las fuentes de información disponibles en un tiempo aceptable. Además, muchas veces una computadora tampoco tiene la memoria necesaria para almacenar los datos que necesita el algoritmo, por lo que hay que recurrir a datos en disco, lo que hace más lento aún el procesamiento. Esta es una situación característica donde se puede intentar resolver el problema explotando el poder de cómputo y la memoria disponible en un conjunto de

procesadores. Así, la paralelización de los algoritmos de detección es la opción para obtener en un tiempo razonable los sucesos de una parte significativa de las noticias existentes en línea.

En este trabajo se expone la paralelización de un algoritmo de agrupamiento incremental que busca los conjuntos compactos existentes en un flujo de noticias en línea. El algoritmo paralelo obtenido logra un adecuado balance de carga entre los procesadores al repartir equitativamente los objetos entre los procesadores y lograr que cada procesador realice aproximadamente el mismo trabajo. Los resultados experimentales obtenidos demuestran la validez de la paralelización realizada.

El K-Means paralelo no es directamente aplicable a la detección de sucesos debido a la naturaleza incremental de este problema. El algoritmo GLC busca de forma incremental las componentes conexas en grafos de semejanzas, por lo que su paralelización podría ser utilizada directamente para detectar los sucesos. Sin embargo, este algoritmo presenta un elevado efecto de encadenamiento, lo que provoca que noticias muy poco relacionadas entre sí se ubiquen en el mismo suceso, por lo que no se considera satisfactorio para esta aplicación. Para paralelizar el compacto incremental supone una arquitectura de computadora paralela con memoria distribuida por lo que se utiliza el paradigma de paso de mensajes. Dentro de este paradigma se utiliza la estrategia de particionado de datos y el esquema de maestro-esclavos. En distintas fases del algoritmo un procesador actuará como maestro y los restantes, como esclavos. En la paralelización del algoritmo compacto reparte uniformemente los datos entre los procesadores de forma tal que el procesador i almacena la descripción del documento j si el resto de la división de j entre p (cantidad de procesadores) es i. Con esto, se trata de lograr un equilibrio de carga entre los procesadores, lo cual es una característica deseable en todo algoritmo paralelo. Además de la descripción de sus documentos, cada procesador mantiene para cada documento el valor de su máxima 0-semejanza y los índices de los documentos conectados con él en el grafo $\max - S$. Cada procesador mantiene también una lista de grupos (conjuntos compactos) y, a su vez, de cada grupo almacena una lista de los documentos del procesador que pertenecen a ese grupo. No necesariamente todos los procesadores contienen a todos los grupos, pues un procesador sólo tiene los grupos a los que pertenecen sus documentos según la distribución de carga realizada.

Ante la llegada de un nuevo documento d, cada procesador calcula la semejanza de sus documentos con d y actualiza la parte del grafo que involucra a sus documentos, lo cual requiere un intercambio de información entre los procesadores. En este proceso se determina qué grupos pueden perder la propiedad de ser compactos. Durante la actualización del grafo, cada procesador construye los conjuntos GruposAProcesar, DocumentosAUnir y

GruposAUnir de forma similar al caso secuencial. Además, cada procesador crea el conjunto ArcosPerdidos que contiene los pares de documentos (d0, d00), donde d0 es un documento de este procesador y d00 es un documento de otro procesador de forma tal que el arco existente entre ellos se rompió. Este rompimiento se produce debido a que es el más 0-semejante a d0 y d00 deja de serlo.

Al finalizar la actualización del grafo de máxima 0-semejanza, se reconstruyen los conjuntos compactos a partir de d y de los documentos que pertenecen a los grupos que pueden perder la propiedad de ser compactos. Los conjuntos compactos que no tienen documentos conectados con d permanecen inalterados. Para ello, cada procesador, a partir de sus documentos que pertenecen a los grupos que pueden perder la propiedad de ser compactos, construye los subconjuntos de compactos que pueden formarse. Para cada uno de estos subconjuntos se forma el conjunto Llegada, que contiene a todos los documentos que están conectados con algún documento de ese subconjunto y no pertenecen a dicho subconjunto.

Nótese que al conjunto Llegada siempre pertenecerán documentos de otro procesador. Luego, el procesador 0 conforma los nuevos conjuntos compactos a partir de estos subconjuntos y sus conjuntos Llegada asociados. Si un subconjunto compacto tiene al menos un documento que pertenece al conjunto Llegada asociado a otro subconjunto compacto, entonces estos dos subconjuntos forman un mismo conjunto compacto.

Aplicando esta propiedad se construyen los conjuntos compactos. Nótese que cuando se unen dos subconjuntos compactos, es necesario formar también un conjunto de llegada.

Los pasos del algoritmo se muestran a continuación:

- ✓ El procesador p0 difunde la descripción del nuevo documento d.
- ✓ Actualización del grafo max − S. En cada procesador pi: Para cada uno de sus documentos se calcula la semejanza con d, se actualiza el valor de su máxima 0-semejanza y el conjunto de los documentos conectados con ´el en el grafo max − S. Se crean los conjuntos GruposAProcesar, DocumentosAUnir y ArcosPerdidos. Cada vez que se incorpora un documento a DocumentosAUnir se elimina del grupo al que pertenecía. Se crea el conjunto GruposAUnir formado inicialmente por los grupos donde existen documentos de este procesador para los cuales d se incorpora a su conjunto de documentos más 0-semejantes. Se crean los conjuntos AEli = conjunto de los documentos de este procesador de los cuales d es el más

0- semejante y DeEli = conjunto de sus documentos que son más 0-semejantes a d. Se calcula además la máxima 0-semejanza de d con los documentos del procesador (MaxSemi) [47].

3. ANÁLISIS DE ALGORITMOS PARALELOS

En el desarrollo de la investigación se estudiaron ocho algoritmos los cuales permiten encontrar itemsets frecuentes en la tarea de asociación y realizando una búsqueda detallada de cada uno de ellos como se muestra en la tabla 1, se logró obtener resultados tanto teóricos como prácticos.

Los resultados teóricos se realizaron aplicando el diseño de algoritmo en paralelo para cada uno de ellos, y en segunda instancia se efectuó pruebas para los algoritmos de Apriori (Bodon) y Fp-Growth (Mahout) realizando pruebas de tiempo de ejecución, con un determinado conjunto de datos de la canasta de mercado.

	ALGORITMOS							
	APRIORI	**COUNT DISTRIBUTION**	**DATA DISTRIBUTION**	**CANDIDATE DISTRIBUTION**	**FP-GROWTH**	**EQUIPASSO**	**CBA**	**AIS**
Utilización de Reglas de Asociación	x	x	X	x	X	x	X	X
Genera Itemsets	x	x	X	x	X	x	x	X
Utilización de Iteraciones	x	x	X	x	X	x		x
Implementación en Paralelo	x	x	X	x	X			

Tabla 1. Comparación de algoritmos

En la tabla 1, se muestra las características importantes que posee un algoritmo en paralelo, se observa que los algoritmos que cumplen en su totalidad las características son Apriori, Count Distribution, Data Distribution, Candidate Distribution y Fp-Growth, teniendo en cuenta que CC, DD y CD son algoritmos que están basados en Apriori.

El algoritmo de EquipAsso y AIS, también cumple con las características aunque en la actualidad se está trabajando para la implementación en paralelo para el algoritmo EquipAsso.

El algoritmo de CBA, solo cumple con dos de las características que son la utilización de reglas y la generación de items.

3.1 PRUEBAS TEÓRICAS

Se efectuó un estudio de análisis de diseño de algoritmos en paralelo aplicando las etapas que se necesitan para ser paralelizado un algoritmo tal y como se muestra en la figura1.

Algoritmo Apriori

Teniendo en cuenta el diseño de algoritmos paralelos se observa que el algoritmo Apriori tiene facilidad en realizar la paralelización debido a que este algoritmo realiza múltiples recorridos por un conjunto determinado de datos.

El algoritmo Apriori permite particionar el conjunto de transacciones, ya que la partición busca subdividir el problema lo más que se pueda, seguidamente se realiza la comunicación del algoritmo teniendo en cuenta dos fases, primero se delimitan los canales que conectan las tareas que requieren los datos con las que lo poseen. Y en el segundo se especifica la información o mensajes que debe ser enviado y recibidos en estos canales.

Para el diseño en paralelo también se realiza la agrupación que va de lo abstracto a lo concreto y se revisa el algoritmo tratando de realizar un algoritmo que se ejecute eficientemente sobre cierta clase de computadoras, reduciendo así el número de mensajes y el costo de comunicación. Finalmente, se realiza la asignación que consiste en determinar en qué procesador se ejecutara cada tarea. A continuación, el procedimiento propuesto por [36] es el siguiente:

1. Contar las ocurrencias de cada ítem en la base de datos para encontrar los -itemsets frecuentes.

2. Particionar el conjunto de los 1-itemsets frecuentes en p partes: {P1, P2, …, p}.

3. Aplicar Apriori o DHP (descrito en un epígrafe posterior) para encontrar todos los itemsets frecuentes hasta la parte procesada, desde la última (Pp) hasta la primera (P1), determinando sus soportes sobre la base de datos. La lógica de este paso es la siguiente: cuando la parte Pp se ha procesado, se han encontrado todos los itemsets frecuentes cuyos ítems están en Pp.

Cuando se procesa la próxima parte Pp-1, se pueden encontrar todos los itemsets frecuentes cuyos ítems están en Pp-1 y Pp, con sus prefijos en Pp-1.

Más específicamente, se determinan los itemsets frecuentes considerando en sus extensiones sólo los itemsets de Pp que son frecuentes. Este procedimiento continúa hasta que se procese P1.

Se asume, sin pérdida de generalidad, que los ítems están ordenados en orden lexicográfico, de forma tal que en las p partes: $\forall\, i < j, \forall\, a \in Pi, \forall\, b \in Pj\ a < b$. Este orden permite conocer si los sub-itemsets son frecuentes por haberse ya procesado partes léxicamente superiores, facilitando la aplicación de la propiedad de clausura descendente.

Algoritmos CD, DD,CD

El algoritmo Apriori es el algoritmo más conocido para el descubrimiento de reglas de asociación, para el que se han propuesto diferentes implementaciones paralelas.

Se presentan tres algoritmos paralelos diferentes llamados Count Distribution (CD), Data Distribution (DD) y Candidate Distribution basados en el algoritmo

Apriori. Agrawal describió tres algoritmos paralelos para obtener todos los itemsets frecuentes y un algoritmo paralelo para la generación de reglas a partir de los itemsets frecuentes. Los algoritmos asumieron una arquitectura de tipo nada-compartido o de memoria distribuida, donde cada uno de los N procesadores tiene memoria y disco propios.

Los procesadores están conectados por una red de comunicación y sólo pueden comunicarse a través de pase de mensajes. Los datos son distribuidos equitativamente en los discos conectados a los procesadores [50].

Estos algoritmos particiona los datos de tal forma que cada procesador puede proceder independientemente. En algún paso *l*, donde *l* es determinado heurísticamente, estos algoritmos dividen los itemsets frecuentes del conjunto L_{l-1} entre los procesadores de tal forma que un procesador P^j puede generar un único C_m^i ($m \geq l$) independiente de los otros procesadores ($C_m^i \cap C_m^j = \varphi$, $i \neq j$).

Al mismo tiempo, los datos son particionados tal que cada procesador puede contar los candidatos en C_m^iindependiente de los demás procesadores.

Lo que hace que cumplan en su totalidad el diseño de algoritmos paralelos iniciando con la partición de los datos, la comunicación y la agrupación de tareas para poder reducir la cantidad de datos a enviar y también reducir el número de mensajes y el costo de la comunicación, para finalmente hacer la asignación que consiste en determinar en qué procesador se ejecutará cada tarea [42].

Algoritmo CBA

Se basa en el algoritmo de minado de reglas de asociación Apriori y calcula todas las reglas que cumplan con los umbrales de soporte y confianza establecidos. Cumple con las cuatro etapas de diseño de algoritmos en paralelo, comenzando por la partición que busca subdividir el problema lo más que se pueda, para observar posteriormente que las tareas se deben efectuar en un mismo número de operaciones de comunicación. Seguidamente realizar la agrupación de tareas, para verificar que la agrupación redujo el costo de comunicación, finalmente se asigna a cada procesador la tarea que va a ser ejecutada.

Algoritmo Fp-Growth

El algoritmo Fp-Growth utiliza una estrategia de recorrido en amplitud y generan conjuntos de items frecuentes.

Fp-Growth se basa en el crecimiento o extensión de los items frecuentes y utiliza una estructura de datos compacta denominada FP-tree, también conocida como árbol de prefijos.

Los 1-itemsets frecuentes se ordenan descendentemente de acuerdo a su soporte, esto permite una mayor compactación de la estructura de datos. La estructura de datos FP-tree es un árbol donde cada nodo se etiqueta con un ítem y su soporte, un Fp-Tree almacena información acerca de conjuntos de ítems frecuentes permitiendo realizar consultas de manera más eficientes. En el segundo recorrido cada transacción se inserta en el árbol, con los ítems ordenados descendentemente de acuerdo a su soporte. De esta forma, los prefijos iguales comparten la misma rama e incrementan en uno el valor de soporte de los ítems que forman la rama.

Según [51], las transacciones de conjuntos de datos suelen compartir ítems frecuentes y, por lo tanto, el tamaño del FP-Tree correspondiente suele ser mucho menor que el conjunto de datos original.

El funcionamiento del algoritmo FP-Growth consiste, justamente, en construir y el FP-Tree que represente los ítems frecuentes de un conjunto de transacciones y, luego, recorrer dicha estructura en busca de los conjuntos de ítems frecuentes.

Este algoritmo se puede paralelizar ya que cumple con las etapas del análisis de algoritmos paralelos iniciando por la partición de los datos buscando subdividir el problema lo más que se pueda, también se logra realizar la comunicación de las tareas que fueron anteriormente particionadas. Luego se realiza una agrupación de las tareas con el fin de reducir la cantidad de datos a enviar, reducir el número de mensajes y el costo de la comunicación, para finalmente determinar en qué procesador se ejecutara la tarea, esto se llama asignación de tareas, logrando así que el algoritmo Fp-Growth sea más rápido y escalable que el algoritmo Apriori.

Algoritmo EquipAsso

Es un algoritmo que se basa en operadores de algebra relacional Associator y EquiKeep. Este algoritmo puede generar el cálculo de conjuntos de ítems frecuentes. Lo que asegura que se puede paralelizar iniciando con la partición de datos y tareas en muchas partes pequeñas, con el objetivo de identificar el máximo de ocurrencias, seguidamente se realiza la dependencia entre tareas, comunicación de datos para establecer los canales, para así poder minimizar el

coste de las comunicaciones, posteriormente se realiza la agrupación de las tareas para mejorar las prestaciones y simplificar la programación con el objetivo de mantener la escalabilidad sin modificar el diseño y también reducir los costes de ingeniería de software. Finalmente asignar tareas a procesadores con el fin de equilibrar la carga y minimizar las tareas conectadas a un mismo procesador.

Algoritmo de AIS

Este algoritmo permite obtener una representación del conocimiento que es útil para caracterizar las regularidades que se pueden encontrar en grandes bases de datos.

Este algoritmo se le puede aplicar el diseño de algoritmos en paralelo ya que se puede particionar el problema en actividades más pequeñas, posteriormente especificar la información o mensajes que deben ser enviados y recibidos a esto se llama comunicación. Seguidamente con la agrupación de tareas se puede reducir la cantidad de datos a enviar, reduciendo así el número de mensajes y el costo de comunicación, seguidamente de realiza la agrupación de tareas que puede ayudar a reducir la cantidad de datos a enviar y así reducir también el número de mensaje, finalmente se determina la asignación de tareas que serán enviadas a cada procesador para ser ejecutadas.

3.2 PRUEBAS DE ALGORITMOS EN SECUENCIAL

Las pruebas que se realizaron fueron para indicar el tiempo de ejecución y la eficiencia de los algoritmos investigados. Las pruebas se plasmaron mediante una herramienta llamada clúster para este caso se utilizó el clúster Tinku, es un modelo de clúster heterogéneo de alto rendimiento, fuertemente acoplado para apoyar y fomentar investigaciones que requieran de gran capacidad de procesamiento de datos, y a la vez eficiencia de tiempo en entrega de resultados, Este clúster tiene un componente principal que es el Frontend que tiene un procesador i7, es el encargado de enviar tareas a los nodos, aquí los usuarios pueden conectarse al sistema, presentar tareas, compilar código y distribuir procesos, el clúster está compuesto por dos nodos, llamados compute-0-1 y compute-0-0, con procesadores i3, estos nodos están conectados entre sí por un canal de comunicación.

Para desarrollar las pruebas de los algoritmos estudiados durante la investigación se obtuvieron resultados tanto para Apriori y Fp-Growth. Para el algoritmo Apriori se utilizó la librería de Bodon y para Fp-Growth se utilizó la librería de Mahout, utilizando un conjunto de 14000 datos de canasta de mercado, son datos que son representados periódicamente por productos comprados durante un periodo de tiempo específico, estos datos fueron

descargados de la página principal dataset *(http://grouplens.org/datasets/movielens/).*

Las pruebas se realizaron haciendo una comparación de los tiempos de ejecución y el tamaño de memoria de cada uno de los algoritmos que utilizaría cada procesador que está incluido en el clúster heterogéneo.

Las pruebas que se efectuaron consistieron en ir aumentando los procesadores con el fin de determinar el rendimiento del tiempo de ejecución y la ocupación de memoria para cada uno los algoritmos tanto para Apriori y Fp-Growth.

También se utilizaron los resultados obtenidos durante las pruebas en la herramienta de Ganglia que fue desarrollada en la Universidad de Berkeley por la división de ciencias.

Para acceder a Ganglia, se abre un navegador, y en la parte de dirección se digita: http*://localhost/ganglia*. Ganglia utiliza un protocolo multicast y un árbol de conexiones punto a punto entre los nodos del clúster, con el fin de indicar cómo está el funcionamiento de cada uno de los procesadores que están en conexión al clúster. El manejo de Ganglia es muy intuitivo, indica la carga actual de cada procesador, el uso de la red, la cantidad de memoria ocupada. Por cada nodo muestra un reporte detallado del estado y gráficas de la actividad reciente.

Ganglia permite al usuario ver de forma remota estadísticas en tiempo real según el procesamiento que se realiza durante la ejecución [49].

3.2.1 Pruebas y resultados de apriori (bodon). Se efectuaron tres experimentos diferentes en la herramienta clúster Tinku con 14000 datos haciendo que para cada prueba se vayan aumentando los procesadores con el fin de observar que a medida que se aumentan los procesadores el tiempo disminuye debido a que el clúster Tinku reparte las cargas para cada uno de los procesadores que están incluidos en él. De este experimento se obtuvo diferentes tiempos por cada procesador, a continuación, en la tabla 2, se indican los tiempos obtenidos por los procesadores.

Prueba 1. En la prueba se ejecutaron los datos en el clúster obteniendo como resultado los siguientes tiempos de ejecución de cada uno de los procesadores. El tiempo de Ejecución de los datos es tomado en segundos. Como se muestra en la tabla 2.

TIEMPOS DE EJECUCIÓN EN EL CLUSTER TINKU	
Procesador	**Tiempo(Segundos)**
Tinku Local	**1.95**
Compute-0-0	**1,55**
Compute-0-1	**1,34**

Tabla 2. Tiempo de ejecución de a priori (bodon)

En la tabla 2, se observa que a medida que van aumentando los procesadores el tiempo de ejecución va disminuyendo, esto es debido a que el clúster comparte las cargas con los procesadores.

En cada prueba se calcula la eficiencia del algoritmo en su totalidad, realizando así los siguientes cálculos: haciendo la suma de todos los tiempos de ejecución por cada uno de los procesadores contenidos que procesaron los datos. Finalmente, se realizó la división de la suma de los tiempos entre los procesadores que se encuentran en el desarrollo de la investigación, así:

$$E = \frac{sp}{p}$$

Donde

$$Sp = \frac{ts}{tp}$$

Ts= tiempo de ejecución del algoritmo secuencial

Tp=Tiempo de ejecución del algoritmo paralelo con p procesadores.

A continuación, la tabla 3, muestra los resultados de eficiencia que se obtuvo del algoritmo Apriori (bodon), la eficiencia permite medir la cantidad de potencia de procesamiento disponible que está siendo usada durante el procesamiento de datos que se obtuvo para los tres procesadores.

Procesadores	Eficiencia
Tinku Local	2,5
Compute-0-0	1,6
Compute-0-1	1,2

Tabla 3. Eficiencia de a priori (bodon)

Se observa en la tabla 3, que la eficiencia va disminuyendo a medida que crecen el número de procesadores, finalmente se define que este algoritmo es eficiente en el procesamiento de datos ya que obtuvo una eficiencia moderada con los tres procesadores.

Seguidamente, la figura 15 muestra la eficiencia que se generó utilizando los tres procesadores que están incluidos en el clúster heterogéneo, de esta figura se concluye que a medida que crece el número de procesadores la eficiencia disminuye.

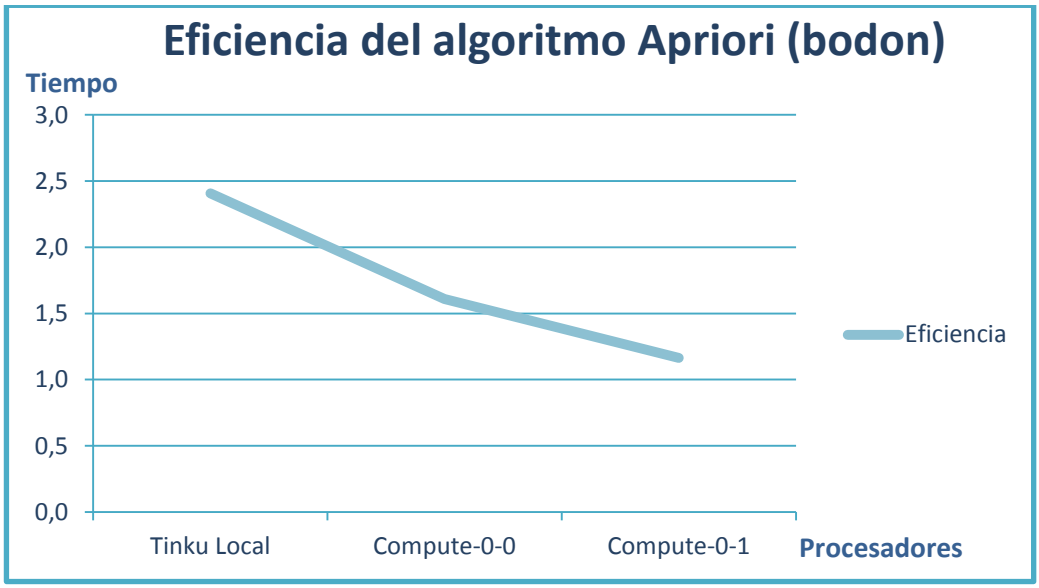

Figura 15. Eficiencia del algoritmo a priori (bodon)

Además, al efectuar las pruebas en el clúster se obtuvo algunas gráficas mediante la utilización de Ganglia, que permite al usuario ver de forma remota estadísticas en tiempo real según el procesamiento que se realiza durante la ejecución del algoritmo Apriori (bodon).

La figura 16, indica el estado de clúster sin ejecutar los algoritmos. En Esta gráfica se observa que el procesador del clúster está estable sin ningún cambio debido a que no se han ejecutado ningún algoritmo.

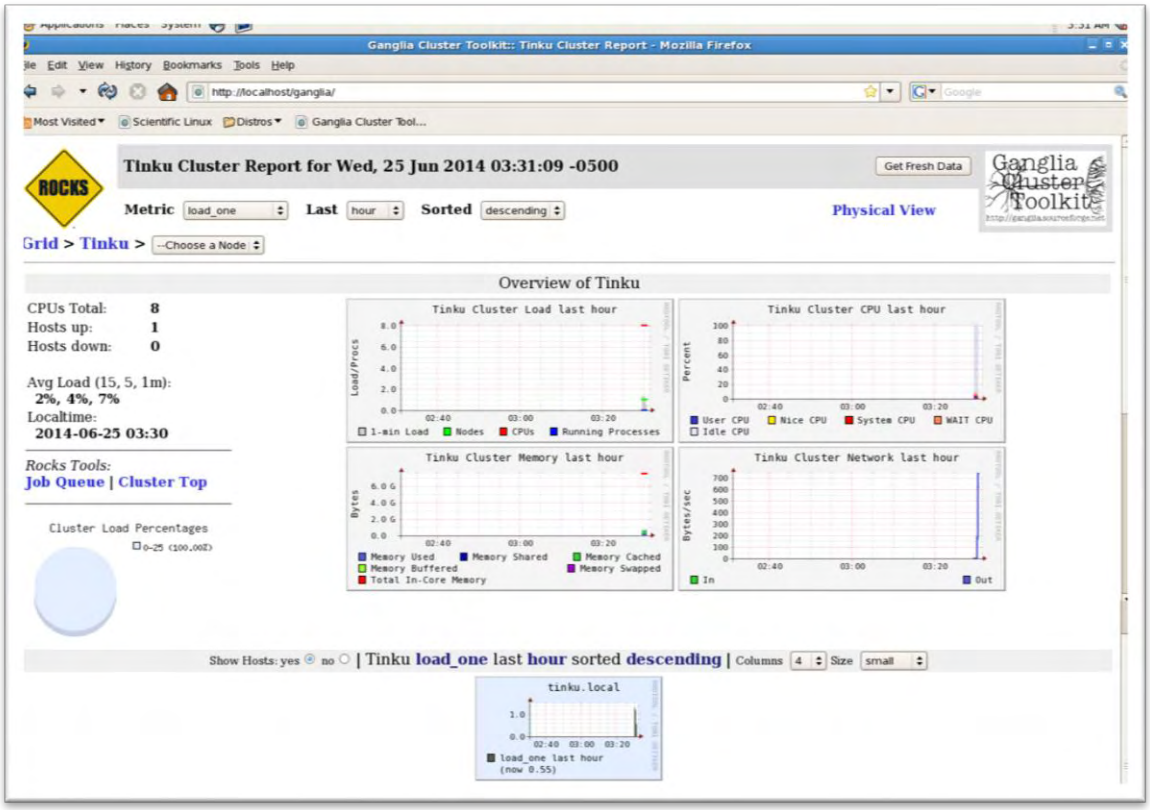

Figura 16. Tinku local

La figura 17, enseña la ejecución del algoritmo Apriori (Bodon) en el servidor, es decir en el Frontend del clúster, aquí se observa que el procesador del clúster cambia según el proceso que se está realizando, también permite observar las características de clúster tanto de la última carga que efectuó, el estado de memoria que está siendo utilizada y la utilización de la red para la conexión de los procesadores. Se muestra que el porcentaje de carga está en un 100% debido a la no conexión de otros procesadores.

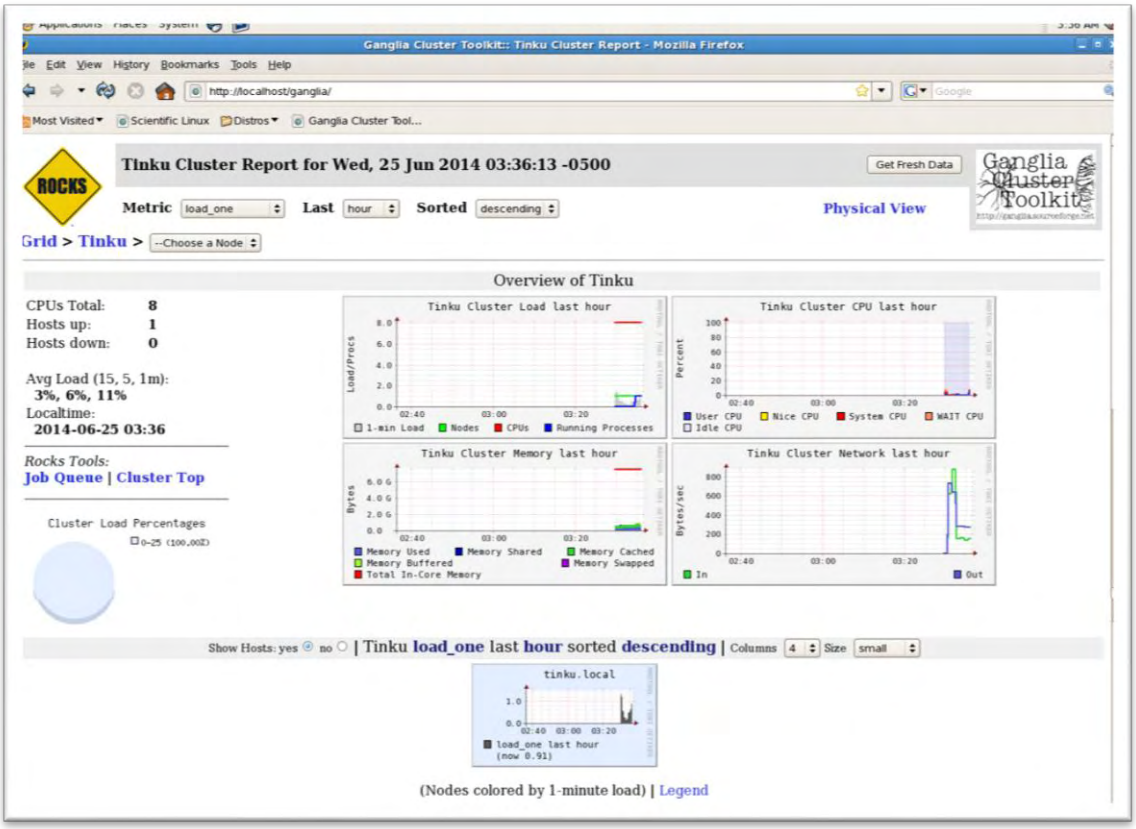

Figura 17. Tinku local en ejecución del algoritmo apriori

La figura 18, muestra los procesos de utilización de métricas de memoria del clúster, en la primera gráfica revela la cantidad de memoria de buffer que utilizó el algoritmo Apriori (bodon) con un valor de 47,992 KB, la segunda gráfica mide la cantidad de memoria de cache tomando un valor de 483,848 KB, la gráfica siguiente hace referencia a la cantidad de memoria disponible, que tiene un valor de 7,052,628 KB, la siguiente gráfica muestra la cantidad de memoria compartida para este caso toma un valor de 0 ya que no hay conexión con los procesadores.

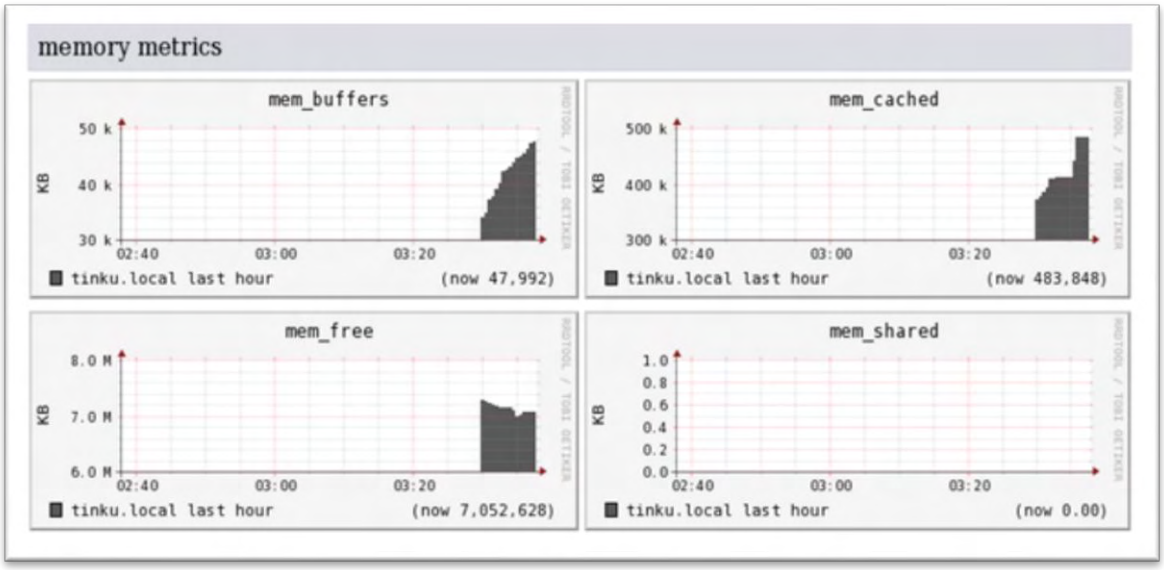

Figura 18. Métricas de memoria tinku local

En la figura 19, indica la ejecución del algoritmo Apriori (bodon) incluyendo el procesador compute-0-1 donde se observa la división de las tareas de los dos procesadores que están siendo utilizadas, en esta gráfica ya se genera la conexión que existe entre el servidor y el procesador compute-0-1.

Figura 19. Procesador tinku local y compute-0-1 en ejecución

La figura 20, muestra los procesos de utilización de métricas de memoria del compute-0-1, en la primera gráfica indica la cantidad de memoria de buffer que utilizó el algoritmo Apriori (Bodon) con un valor de 12,908 KB, la segunda gráfica es la que mide la cantidad de memoria de cache tomando el valor de 178,220 KB, la gráfica tercera hace referencia a cantidad de memoria disponible que posee el procesador, para este caso toma el valor de 786,868 KB, y la última gráfica muestra la cantidad de memoria compartida tomando un valor de 81KB.

Figura 20. Métrica de memoria de compute-0-1

La figura 21, plasma la ejecución del algoritmo Apriori (Bodon), con tres procesadores, ahora incluyendo el procesador de compute-0-0 donde indica un mayor procesamiento, aunque el tiempo que tardo en procesar fue menor, debido al aumento de los procesadores el tiempo de ejecución fue descendiendo de acuerdo a los procesadores que estaban incluidos en el clúster, esta gráfica muestra que aumento la carga y la memoria que utilizó en la ejecución del algoritmo Apriori (Bodon).

Figura 21. Tinku local, compute-0-1 y compute-0-0 en ejecución de apriori

La figura 22, enseña los procesos de utilización de métricas de memoria del compute-0-0, en la primera gráfica indica la cantidad de memoria de buffer que utilizó el algoritmo Apriori (bodon) con un valor de 58,680 KB, la segunda gráfica es la que mide la cantidad de memoria de cache que toma un valor de 604,684 KB, la gráfica siguiente hace referencia a cantidad de memoria disponible que posee un valor de 6,660,400 KB, la última gráfica muestra la cantidad de memoria compartida tomando un valor de 345KB.

Figura 22. Métricas de memoria de compute0-0

Al finalizar las pruebas con los procesadores se obtuvo un archivo de texto el cual indica las Frecuencias y ocurrencias de Itemsets que se adquirieron en la ejecución del algoritmo Apriori (Bodon) en el procesamiento de los 14000 datos, se consiguió un resultado de 289 ocurrencias.

El archivo de texto que se obtuvo durante la ejecución de algoritmo se generó sobre conjunto de elementos frecuentes, utilizando un valor umbral, este valor se incluye a la hora de ejecutar dicho algoritmo, tomando un valor de 0.02.

3.2.2 Pruebas y resultados de fp-growth (mahout). Se efectuaron tres experimentos diferentes con la herramienta clúster Tinku aumentando en cada experimento un procesador para la ejecución del algoritmo Fp-Growth, logrando así como resultado la cantidad de items que se generan durante la ejecución mostrando como recomendado los ítems generados durante el procesamiento de los datos. Estos resultados varían diminutivamente de acuerdo a la cantidad de datos que se están procesando. También este algoritmo permite obtener el coeficiente de correlación que mide en una escala de 0 a 1, tanto en dirección positiva como negativa. Un valor de "0" indica que no hay relación lineal entre las variables. Un valor de "1" o "–1" indica, respectivamente, una correlación positiva perfecta o negativa perfecta entre dos variables. Este resultado depende de la cantidad de datos que procesa el algoritmo entre más datos se obtenga el coeficiente de correlación es más eficiente.

3.2.3 Prueba: en la prueba se ejecutaron los datos en el clúster obteniendo como resultado los siguientes tiempos de ejecución de cada procesador, estos tiempos son menores a los obtenidos en el algoritmo Apriori (Bodon), como se muestra en la tabla 4.

Tiempos de ejecución en el cluster Tinku	
Procesador	**Tiempo(Segundos)**
Tinku Local	**1,67**
Compute-0-0	**1,2**
Compute-0-1	**1,15**

Tabla 4. Tiempo de ejecución de Fp-Growth (mahout)

Para calcular la eficiencia del algoritmo en su totalidad se realizó los siguientes cálculos: realizando la suma de todos los tiempos que se procesaron todos los datos. Para finalmente realizar una división de la suma de los tiempos entre los procesadores que se encuentran en el desarrollo de la investigación así:

$$E = \frac{sp}{p}$$

Donde

$$Sp = \frac{ts}{tp}$$

Ts= tiempo de ejecución del algoritmo secuencial

Tp=Tiempo de ejecución del algoritmo paralelo con p procesadores.

A continuación, se muestra en la tabla 5 los resultados de eficiencia, que permite medir la cantidad de potencia de procesamiento disponible que está siendo usada durante el procesamiento de datos que se obtuvieron para los tres procesadores.

Procesadores	Eficiencia
Tinku Local	2,4
Compute-0-0	1,7
Compute-0-1	1,2

Tabla 5. Eficiencia de Fp-Growth (mahout)

La tabla ,5 indica que la eficiencia va disminuyendo a medida que aumenta el número de procesadores, en conclusión se define que el algoritmo Fp-Growth (Mahout) es eficiente en el procesamiento de datos ya que cumple con el objetivo esperado durante la ejecución de los datos.

En la figura 23, se observa que el algoritmo Fp-Growth (Mahout) tiene un buen rendimiento en cuanto a la eficiencia debida al tiempo de ejecución de cada uno de los procesadores.

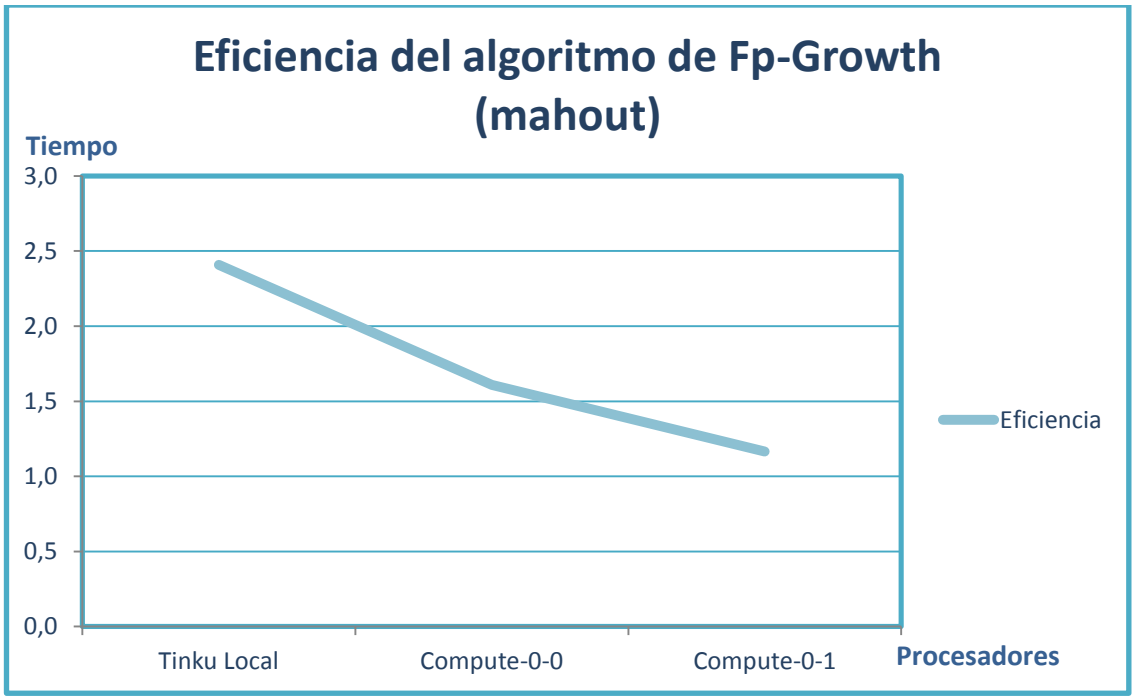

Figura 23. Resultado de eficiencia Fp.Growth (mahout)

Se obtuvo pruebas de la herramienta Ganglia tanto del tiempo real, el movimiento del clúster, el tamaño de la carga, la memoria disponible y la conexión de los nodos, como se observa en la figura 24.

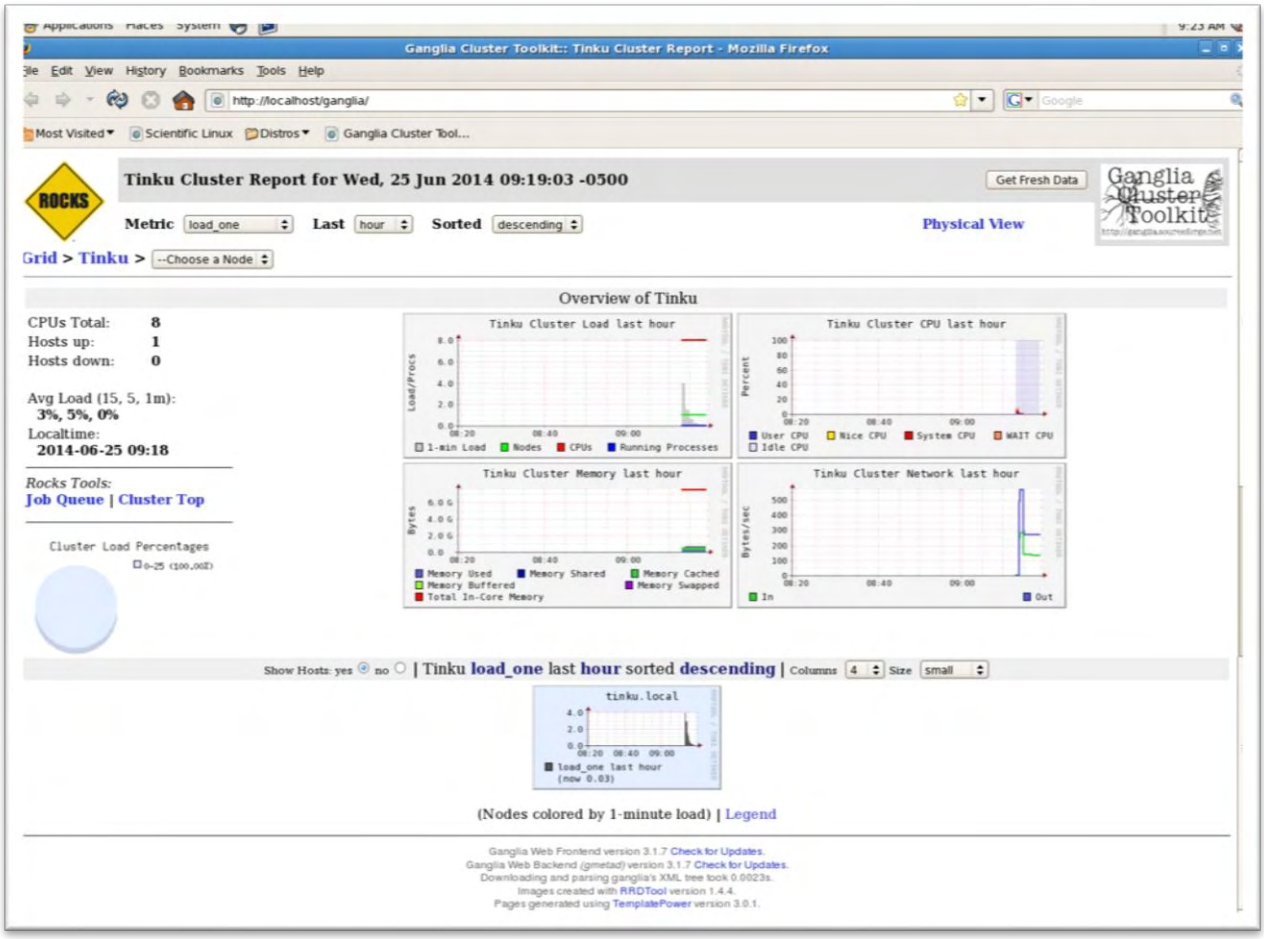

Figura 24. Tinku local sin ejecución Fp-Growth (mahout)

La figura 25, indica los procesos de utilización de métricas de memoria del clúster es decir del servidor, en la primera gráfica indica la cantidad de memoria de buffer que utilizó el algoritmo Fp-Growth (mahout) con un valor de 58,680 KB, la segunda gráfica mide la cantidad de memoria de cache que toma un valor de 604,684 KB, la gráfica siguiente hace referencia a cantidad de memoria disponible que tiene un valor de 6,660,400 KB, la siguiente gráfica muestra la cantidad de memoria compartida para este caso toma un valor de 0 ya que no hay conexión con los procesadores.

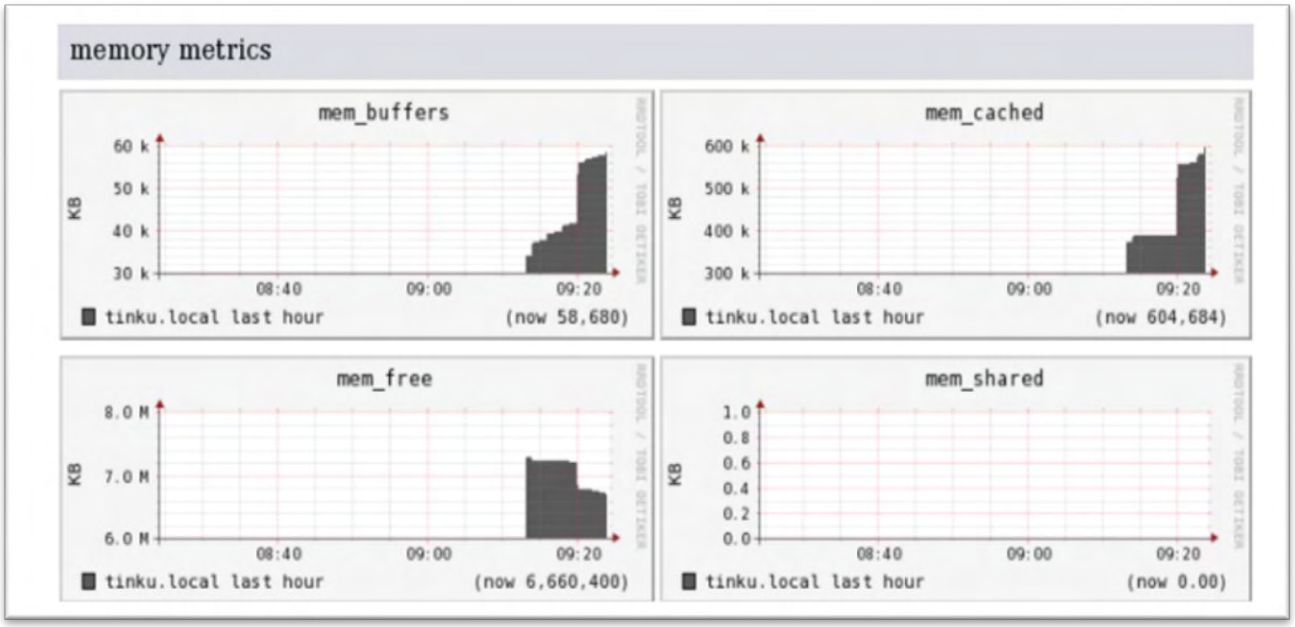

Figura 25. Métricas de memoria de Fp-Growth (Mahout)

A continuación, en la figura 26 muestra, el clúster donde se ejecutó el algoritmo de Fp-Growth (Mahout) con un procesador llamado compute-0-1, aquí se observa que antes de ejecutar el algoritmo los procesadores están inactivos es decir no tienen suficiente movimiento debido a que no hay ningún proceso que exponer.

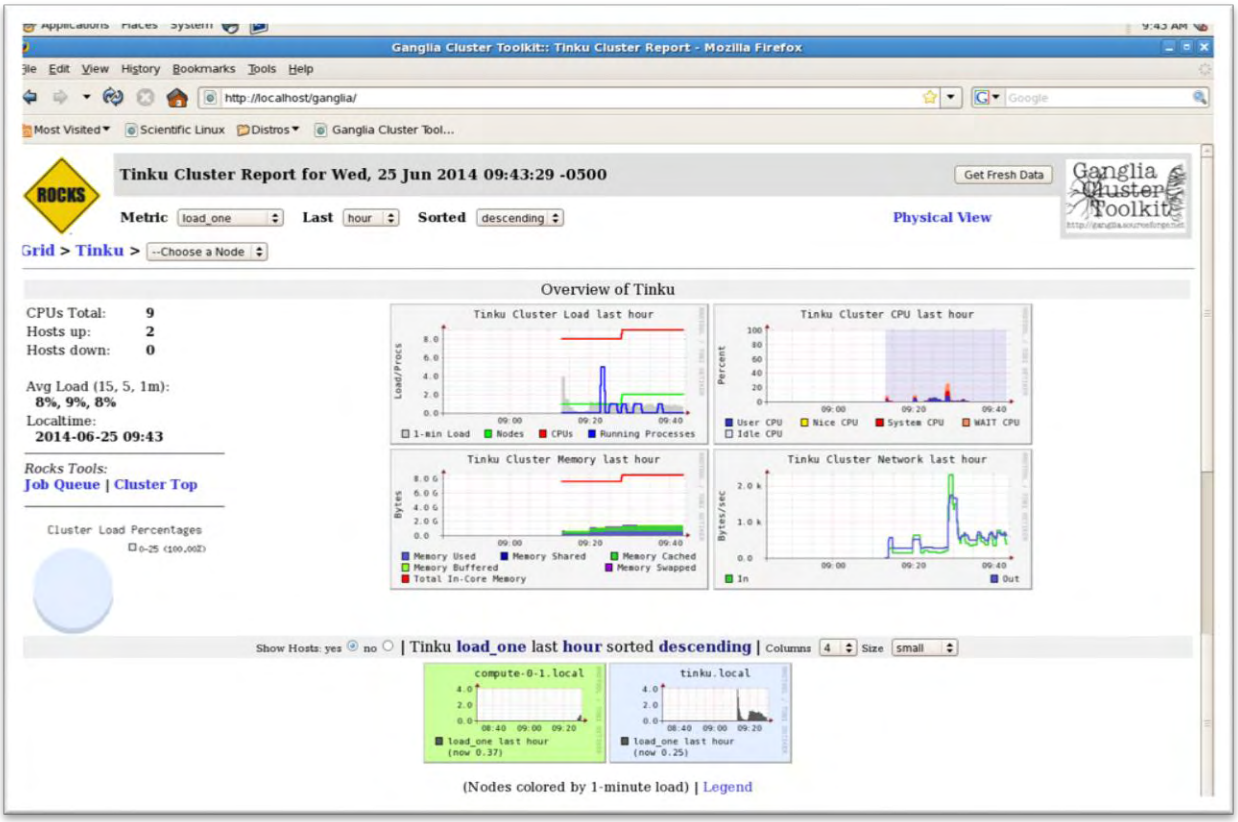

Figura 26. Tinku local y compute-0-1sin ejecutar el algoritmo mahout

Seguidamente, la figura 27 indica que al ejecutar el algoritmo en los dos procesadores se reparten las cargas entre los procesadores que están incluidos en el clúster. Esta gráfica también enseña los movimientos de memoria, carga y la red de conexión de los procesadores que están incluidos en el clúster.

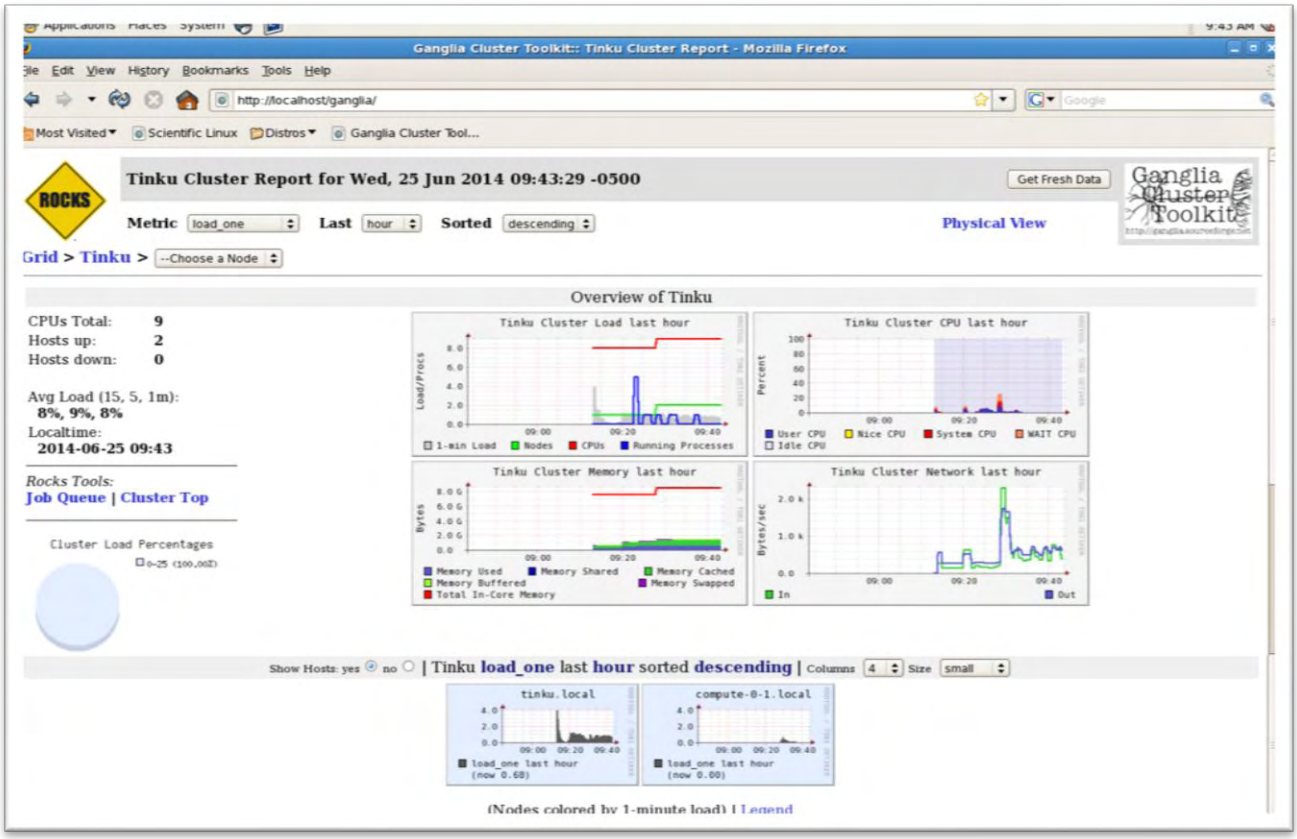

Figura 27. Tunku local y compute-0-1 con ejecución del algoritmo mahout

La figura 28, enseña los procesos de utilización de métricas de memoria del procesador compute-0-1, en la primera gráfica indica la cantidad de memoria de buffer que utilizó el algoritmo Fp-Growth (Mahout) con un valor de 12,816 KB, la segunda gráfica mide la cantidad de memoria de cache que toma un valor de 178,212 KB, la gráfica siguiente hace referencia a cantidad de memoria disponible que tiene un valor de 786,988 KB, la siguiente gráfica muestra la cantidad de memoria compartida para este caso toma un valor de 80.00 KB.

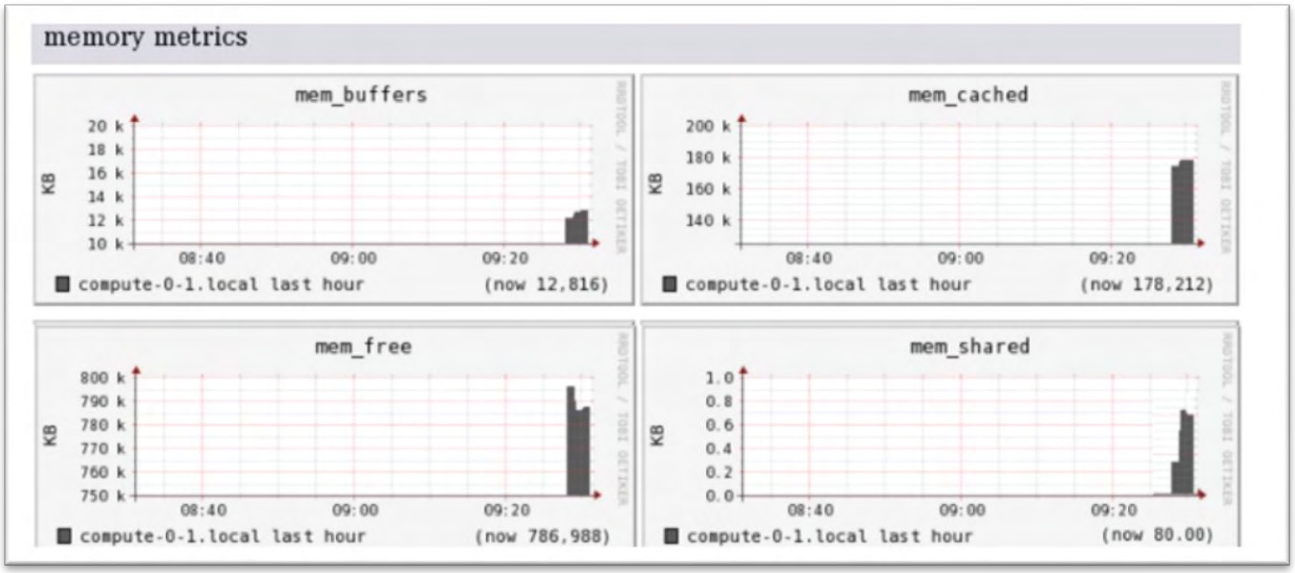

Figura 28. Métricas de memoria de compute-0-1

En la figura 29, está incluido el procesador compute-0-0, se observa que cada procesador está ejecutado según las tareas que se está procesando en el algoritmo de Fp-Growth, distribuyendo la carga para cada procesador.

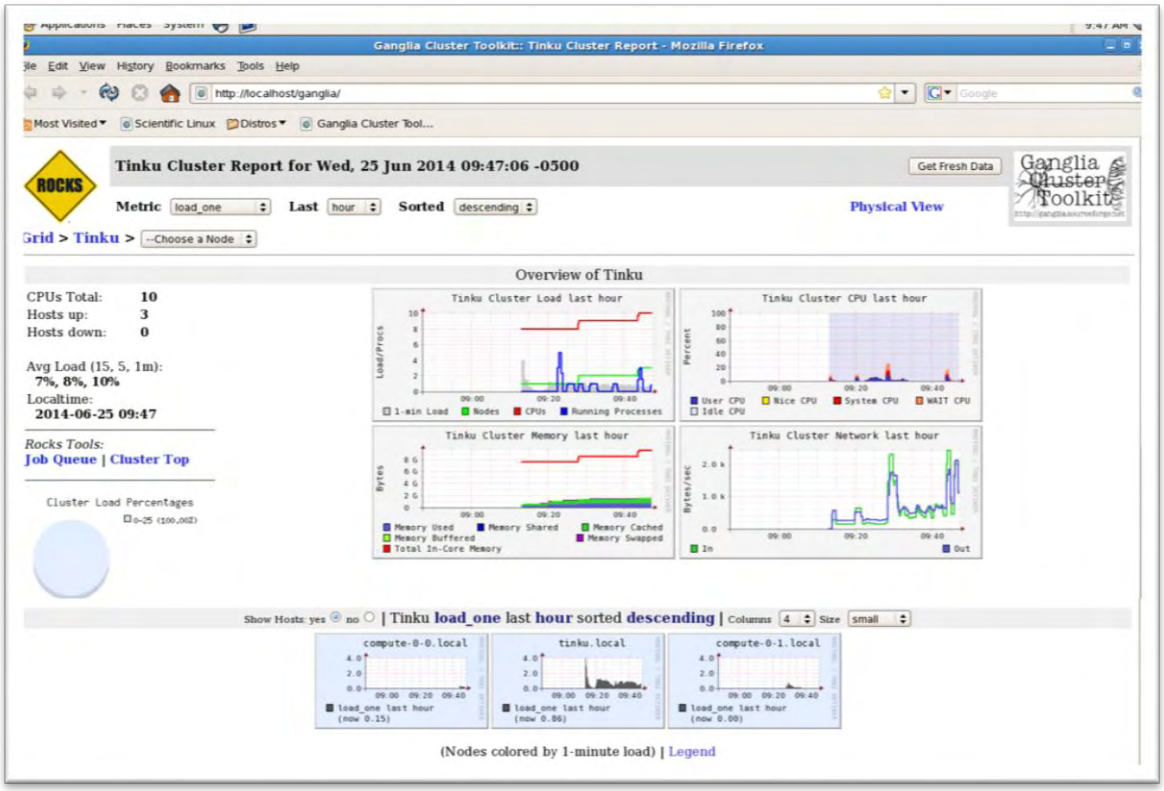

Figura 29. Tinku local, compute-0-1 y compute-0-0 en ejecución del algoritmo mahout

La figura 30, indica los procesos de utilización de métricas de memoria del procesador compute-0-0, en la primera gráfica indica la cantidad de memoria de buffer que utilizó el algoritmo Fp-Growth (Mahout) con un valor de 12,940 KB, la segunda gráfica mide la cantidad de memoria de cache que toma un valor de 178,176 KB, la gráfica siguiente hace referencia a cantidad de memoria disponible que tiene un valor de 786,988 KB, la siguiente gráfica muestra la cantidad de memoria compartida para este caso toma un valor de 1,500 KB.

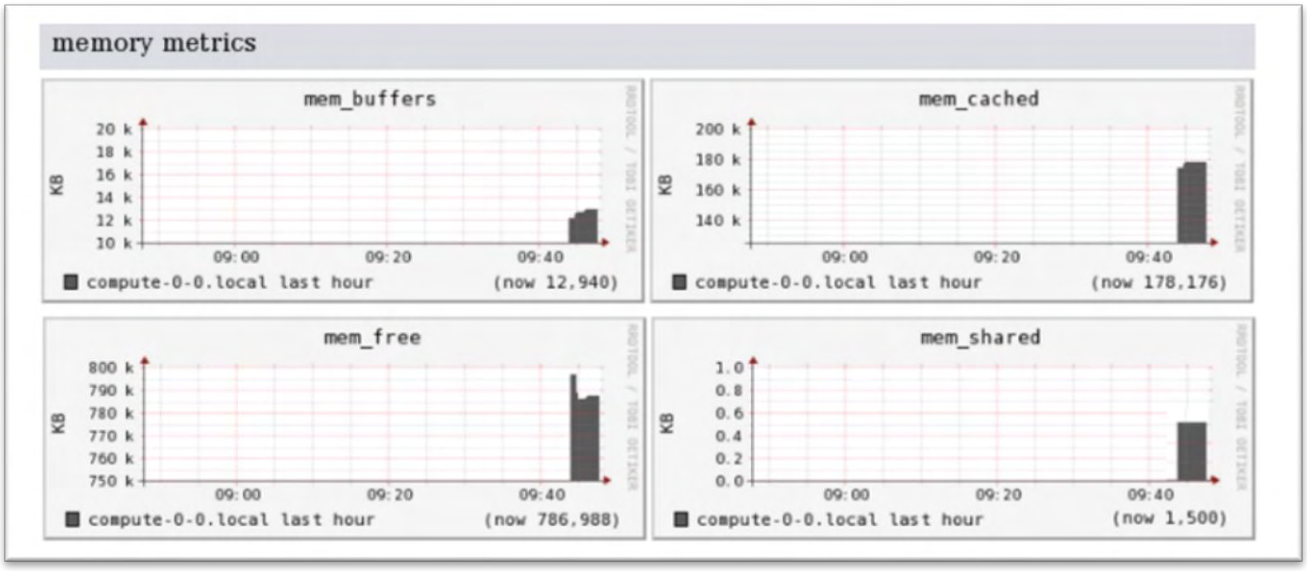

Figura 30. Métricas de memoria de compute-0-0

Después de efectuar las pruebas de cada procesador se muestra un resultado que genera el algoritmo Fp-Growth (mahout) indicando un resumen de la cantidad de datos que se procesaron, aunque el resultado depende de la cantidad de datos que el usuario quiera imprimir en la consola de netbeans.

También este algoritmo muestra un porcentaje de ejecución que hace que a medida que vayan aumentando los datos, la respuesta sea más confiable ya que este valor depende de la cantidad de datos que se procesen entre más datos se tenga mayor en la confiabilidad.

Finalmente, después de haber implementado el algoritmo Apriori (bodon) y el algoritmo Fp-Growth (mahout) en el clúster Heterogéneo, se concluye que es importante implementarlos en el clúster, ya que permite que sean más eficiente y agiles a la hora de ser ejecutados.

Se determina que el algoritmo que mejor rendimiento obtiene en cuanto al tiempo de ejecución y de memoria es el algoritmo de Fp-Growth (mahout), se observa que Fp-Growth (mahout) utilizó menos tiempo de ejecución y los resultados los generan en consola, a diferencia de Apriori (bodon), genera los resultados en archivo de texto, aunque Apriori muestra un resultado más detallado de los itemsets frecuentes que se generaron dentro de la ejecución. Apriori (bodon) también es el más eficiente en cuanto a la respuesta de ejecución ya que es más legible, los resultados obtenidos de la ejecución de Apriori se pueden observar en el CD en la carpeta *Algoritmo Apriori (bodon)*/ resutadof. Debido a que el archivo generado es muy pesado y no se puede imprimir, pero si se lo puede observar con un editor de texto.

Después de realizar las pruebas necesarias y obteniendo los resultados de los dos algoritmos, se determina que el algoritmo más adecuado en cuanto al tiempo de ejecución es Fp-Growth (mahout) debido a que este algoritmo contiene implementaciones de algoritmos de clasificación, agrupación y filtrado colaborativo que tiene soporte para almacenar su modelo en una base de datos, haciendo que Fp-Growth mahout) sea más eficiente, aunque la diferencia con Apriori (bodon) es mínima.

Obteniendo los resultados generados durante las pruebas de ejecución, se observó que el algoritmo Fp-Growth (mahout), es el algoritmo que utiliza más memoria en el procesamiento de los datos, debido al proceso que se efectúa durante la ejecución, ya que Fp-Growth es generado mediante un lenguaje de programación eclipse, a diferencia de Apriori (bodon) que es ejecutado mediante un.exe.

CONCLUSIONES

Finalmente se concluye que el diseño de algoritmos en paralelo, se puede aplicar a los algoritmos que generan itemsets frecuentes, permitiendo así aplicar las cuatro etapas, iniciando por la partición del problema, hacer la comunicación entre canales para enviar los mensajes, seguidamente realizar la agrupación de las tareas para posteriormente hacer la asignación de las tareas a cada procesador.

En el proceso de recolección de información, se concluye que existen varios algoritmos paralelos que ya están implementados, las pruebas realizadas para los algoritmos Apriori y Fp-Growth se realizaron en forma secuencial implementadas en un clúster, con la finalidad de que el rendimiento de ejecución sea más eficiente, a diferencia de procesadores o máquinas personales.

El algoritmo de Fp-Growth (mahout) al ejecutar 14000 datos, tarda un tiempo de 1.15 segundos en el clúster para generar la respuesta, a diferencia del algoritmo Apriori (bodon) procesa en un tiempo de 1.24 segundos para crear el archivo de texto, finalmente se concluye que el algoritmo que mejor rendimiento tiene es Fp-Growth (mahout).

Al realizar las pruebas de los algoritmos es eficiente utilizar el clúster, ya que este permite repartir las cargas entre los procesadores que están incluidos en él. Haciendo eficaz la respuesta de ejecución de los datos de cada uno de los algoritmos implementados.

En el proceso de investigación de los algoritmos de Fp-Growth y Apriori se establece que los algoritmos utilizan elementos frecuentes de una base de datos transaccional, son datos representados periódicamente por productos comprados durante un periodo de tiempo específico, tratando de examinar la estructura, la aplicación y características de los datos, estos algoritmos igualmente generan conjuntos de elementos frecuentes como se constituyen en la minería de datos.

RECOMENDACIONES

Optar que los datos de prueba, tengan un formato de acuerdo con el algoritmo que se desea ejecutar, ya que el algoritmo Apriori sólo necesitaría una identificación de usuario o id usuario. Para el algoritmo de Fp-Grotwh, se requeriría un identificador de usuario, un ítem ID y un valor; los dos algoritmos necesitarán datos que sean de la canasta de mercado.

Implementar un clúster, para que haya mayor eficiencia, en cuanto al procesamiento de datos, ya que para Apriori se lo implementó en un sistema operativo Windows con procesador i3 y se obtuvo una mala experiencia, debido a que el tiempo de ejecución del mismo fue de 43 minutos y al esperar el archivo de texto no se logró abrir, por ser un archivo muy pesado.

REFERENCIAS BIBLIOGRAFICAS

[1] Algoritmo.
García Y. Algoritmos, Universidad Autónoma del Estado de Hidalgo, Enero 2012.

[2]Análisis.
http://www.mastermagazine.info/termino/3840.php

[3] Base De Datos Distribuidas.
http://alarcos.esi.uclm.es/doc/bbddavanzadas/08-09/presentacionDistribucion1.PDF

[4] Definición de Data Warehouse
http://www.inf.udec.cl/~revista/ediciones/edicion3/cwolff.PDF

[5] Definición de Granuralidad.

Arquitecturas que soportan la concurrencia CDI.

[6] Grafos, Universidad Técnica Federico Santa María, Departamento de Electrónica, Junio 2010.

[7] Definición de Items.
http://www.wordreference.com/definicion/%C3%ADtems.

[8]Definición de Lexicográfica.
Garrido A., Montesa S., La Definición de Lexicográfica, Selección y Modificación, Universidad de Malaga.

[9] Definición de Minería de Datos.
Dante C., Martínez J., Reglas de Asociación en Series Temporales: Panorama Referencial y Tendencias, Escuela de Ingeniería de sistemas, Universidad de los Andes Mérida, Venezuela.

[10]Definición de Overhead.
Sistemas Operativos UTN- FRM, Introducción a los Sistemas Operativos.

[11]Definición de Reglas de Asociación http://ccc.inaoep.mx/~emorales/Cursos/NvoAprend/node18.html .

[12] Naiouf M. Procesamiento Paralelo. Balance de carga Dinámico en algoritmos de Storing, Universidad de Nacional de la Plata, Facultad de Ciencias Exactas, Junio 2004.

[13] Definición de Transacción.

Olarte C. BDII, Transacciones.

[14] Hernández J. Estado Actual de la aplicación de la Computación Paralela a la Minería de Datos, Centro de Aplicaciones de Tecnologías de Avanzada, Noviembre 2014.

[15] Minería de Datos, Teleprocesos y sistemas Distribuidos, Licenciatura en Sistemas de Información, FACENA –UNNE, Octubre 2003.

[16] Cruz M. Conceptos Básicos de Base de Datos, mcruz@uaem.mx.

[17]http://mineriadatos.blogspot.com/2009/04/conocimientos-en-bases-de-datos-kdd.html.

[18] Calderón Méndez N. Minería de Datos una Herramienta para la Toma de Decisiones, Universidad de San Carlos de Guatemala Facultad de Ingeniería, Guatemala, Abril de 2006.

[19] Reyes, Vicente. Procesamiento Paralelo en Redes Linux Utilizando MPI. 2003. http://beta.redes-linux.com/manuales/cluster/mpi-spanish.pdf, 10 Junio de 2010.

[19] Flores G. Algoritmos Secuenciales y Paralelos para la Resolución del Problema Inverso de Valores Singulares, Universidad Politécnica de Valencia España, Noviembre 2005.

[20] Gr3-Mel-Alo-JC-Will. Diseño de Algoritmos Paralelos, Instituto Tecnológico de Costa Rica, Octubre 2010.

[21] Hoeger H. Introducción a la Computación Paralela, Centro Nacional de Cálculo Científico Universidad de Los Andes, Mérida – Venezuela.

[22] Claver J. Algoritmos Paralelos y Distribuidos para Resolver Ecuaciones Matriciales de Lyapunov en Problemas de Reducción de Modelos, Universidad Politécnica de Valencia Departamento de Sistemas Informáticos y Computación Algoritmos, Valencia Junio 1998.

[23] Evaluación de Algoritmos Paralelos, Confederación Económica de la Provincia de Buenos Aires.

[24] Gonzáles A., Llanos D., Orden D., Palop B., Ejecución Paralela de Algoritmos Incrementales Aleatorizados, Departamento de Informática, Universidad de Valladolid, España.

[25] Técnicas de Paralelización Automática, Capitulo 2.

[26] Castro I. Paralelización de Algoritmos de Optimización Basados en Poblaciones por Medio de GPGPU, Instituto Tecnológico de la Paz División de Estudios de Posgrado E Investigación Maestría en Sistemas Computacionales, México Julio 2011.

[26] Parte 2. Sistemas Paralelos, Introducción a las Arquitecturas Paralelas, sistemas de Multiprocesamiento, Computadores de Altas Prestaciones, Departamento Tecnología Electrónica.

[27]García F., Programación en paralelo y Técnicas Algorítmicas, Ciencias y Tecnología.

[28] Capítulo 3. Divide y Vencerás, Técnicas de Diseño de Algoritmos.

[29]Solar M., Técnicas de Programación, Universidad Técnica Federico Santa María, Departamento de Informática, 2008.

[30] Claver J., Algoritmos Paralelos y Distribuidos para Resolver Ecuaciones Matriciales de Lyapunov en Problemas de Reducción de Modelos, Universidad Politécnica de Valencia Departamento de Sistemas Informáticos y Computación, Valencia Junio de 1998.

[31] García E., Romero C., Castro C., Ventura S., Usando Minería de Datos para la Continua mejora de cursos de E-Learnig, Escuela Politécnica Superior, Universidad de Córdoba, 2006 España.

[32] Giraldo J. Caracterización de Algunas Técnicas Algorítmicas de la Inteligencia Artificial para el Descubrimiento de Asociaciones entre Variables y su Aplicación en un Caso de Investigación Específico, Universidad Nacional de Colombia, Facultad de Minas Escuela de Ingenierías Medellín 2009.

[33] Trefftz, Christian. Procesamiento Paralelo en EAFIT. 1998. Colombia.

http://www1.eafit.edu.co/drupal/?q=node/549

[34] Rodríguez A., Martínez J., Ariel J., Ruiz J., Minería de Reglas de Asociación sobre Datos Mezclados, Coordinación de Ciencias Computacionales INAOE, México, 31 de Marzo del 2009.

[35] Análisis de Asociaciones, Universidad del Valle.

[36] Medina J., Hernández J., Hernández R., Pérez A., Echevarría A., Gazapo R. Generación de Conjuntos de ítems y Reglas de Asociación. Centro de Aplicaciones de Tecnologías de Avanzada, Mayo 2007.

[37] Villavicencio A. Búsqueda de Patrones Frecuentes en un Conjunto de Datos Mediante el Algoritmo FPGrowth, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile.

[38] Timaran R. EquipAsso: Un Algoritmo para el Descubrimiento de Conjuntos de Ítems Frecuentes sin generación de Candidatos , Facultad de Ciencias de Ingeniería, Universidad de Manizales, Diciembre 2012.

[39]Timaran R., Calderón A., Ramírez I., Alvarado J., Guevara F., Análisis de Desempeño de Equipasso: Un Algoritmo para el Cálculo de Itemsets Frecuentes Basado en Operadores Algebraicos Relacionales, Universidad de Nariño, Departamento de Ingeniería de Sistemas.

[40] Reglas de Asociación, Capitulo 2. http://www.tdx.cat/bitstream/handle/10803/5983/08Eap08de14.pdf?sequence=8

[41] Medina J., Hernández J., Hernández R. Pérez A., Gonzales R., Generación de Conjuntos de Items y Reglas de Asociación, Centro de Aplicaciones de Tecnologías Avanzada, Mayo 2007.

[42] Amalgro J., Hernández _J., Algoritmos Paralelos para el Descubrimiento de Secuencias Frecuentes, cerradas y Maximales, Centro de Aplicaciones de Tecnologías de Avanzada, Abril 2010.

[43] https://www.ibm.com/developerworks/br/java/library/j-mahout/.

[44] Ferenc Bodon, Resultados Sorprendentes de Algoritmos basados en la FIM Trie, Departamento de Ciencias de la Computación y Teoría de la Información, Budapest Universidad de Tecnología y Economía.

[45] Aguilar Cornejo M., Quiroz F., Román A., Jiménez A., Verificación Formal de un Algoritmo de Procesamiento Paralelo de Imágenes Médicas, Depto. Ing. Eléctrica, Área de Computación y Sistemas, División de CBI, UAM-I, 2007.

[46] Nesmachnow S., Algoritmos Genéticos Paralelos y su aplicación al Diseño de Redes de Comunicaciones Confiables, Instituto de Computación,Facultad de Ingeniería Universidad de la República Montevideo, Uruguay, 2004.

[47] Gil-García Etal, Algoritmo paralelo para detectar sucesos en noticias en línea, Septiembre 2003.

[48] Bodon F., Surprising results of trie-based FIM algorithms, Department of Computer Science and Information Theory, Budapest University of Technology and Economics.

[49] Mora Arrellano J., Vinueza Montenegro E., Tinku – Implementación En La Universidad De Nariño, De Un Clúster Heterogéneo De Alto Rendimiento, Fuertemente Acoplado, Departamento de Ingeniería de Sistemas, Universidad de Nariño, San Juan de Pasto, 2010.

[50] Medina J., Hernández J., Pérez A., Hechavarría A., Gonzales R., Generación de Conjuntos de Items y Reglas de Asociación, Centro de aplicaciones de tecnologías avanzadas, 2007.

[51]Pinho J., Metodos de Clasificacion Basados en Asociación Aplicados a Sistemas de Recomendación, Universidad de Salamanca, 2010