UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

REPARACIÓN DE UNA BASE DE DATOS REPLICADA A TRAVÉS DE UNA TÉCNICA DE RECONCILIACIÓN DE CONJUNTOS

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A:

ROGELIO MONTERO CAMPOS

DIRECTOR DE TESIS:
DR JAVIER GARCÍA GARCÍA

2010

FACULTAD DE CIENCIAS
UNAM

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Hoja de datos del jurado:

1.Datos del alumno

Apellido paterno	Montero
Apellido materno	Campos
Nombre(s)	Rogelio
Teléfono	19 99 08 95
Universidad	Universidad Nacional Autónoma de México
Facultad	Facultad de Ciencias
Carrera	Ciencias de la Computación
Número de cuenta	0999002774

2. Datos del tutor

Grado	Dr
Nombre(s)	Javier
Apellido paterno	García
Apellido materno	Gracía

3. Datos del sinodal 1

Grado	Dra
Nombre(s)	Amparo
Apellido paterno	López
Apellido materno	Gaona

4. Datos del sinodal 2

Grado	M en C
Nombre(s)	Egar Arturo
Apellido paterno	García
Apellido materno	Cárdenas

5. Datos del sinodal 3

Grado	L en C C
Nombre(s)	Manuel Alberto
Apellido paterno	Sugawara
Apellido materno	Muro

6. Datos del sinodal 4

Grado	M en I
Nombre(s)	Rene Alejandro
Apellido paterno	Villeda
Apellido materno	Ruz

7.Datos del trabajo escrito.

Título	Reparación de una base de datos replicada a través de una técnica de reconciliación de conjuntos.
Número de páginas	69 p
Año	2010

Agradecimientos

Con mucho cariño a mis padres Angelina y Rogelio por inculcarme el valor de la educación, por apoyarme y darme más de lo necesario en esta etapa de mi vida, este logro es de los tres. A mis hermanas Araceli y Edith asi como a Gustavo, por darme siempre su apoyo, cariño y hermandad. Para todos mis abuelos, primos y tios que me han enseñado a valorar la importancia de la gran familia que tengo. En particular a mis sobrinos por recordarme cuantas cosas he pasado para llegar hasta aquí, esperando que tengan la misma oportunidad y la aprovechen de buena manera.

Gracias al Dr Javier García García por brindarme la oportunidad de realizar éste trabajo bajo su tutela, por involucrarme en la vida académica, por confiar en mi, por sus consejos, ánimos y por compartir sus experiencias profesionales, lo valoro mucho y le estaré siempre agradecido. Así mismo a mis sinodales por sus valiosos comentarios y puntos de vista sobre mi trabajo. Un agradecimiento especial al Dr Humberto Carrillo Calvet y a Carlos Carrillo por las facilidades técnicas y de espacio para llevar a cabo parte de este trabajo.

Para mis hermanos César y Mauricio que me han confiado más que su amistad, que siempre han estado, y estarán, en los mejores y peores momentos junto a mi. A César A., Diego, Eder, Edith, Jesus, Laura, Luis, Manuel, Mario, Rafa, Tere, Tirado, Pilar, Vianey y Yeudiel por tantos años de amistad en los que hemos estado juntos pasando buenos momentos, sin éllos hubiera sido muy difícil terminar mi carrera. También a mis amigos de la facultad Adri, Armando, Cavazos, Christian, Daniel, Felipe, Israel, Ivan, José, Luis, Omar D., Omar, Oswaldo, Pame, Rebe, Robert, Sandrita, Xoch, Victor y a Vicky por acompañarme en el 'sufrimiento' y donde sin duda nos la pasamos muy bien, de manera especial a mi hermano gurú Rodrigo, a mi querida amiga Angelita y a los SoundException();.

Por último a la Universidad Nacional que me ha brindado tantas oportunidades y me ha permitido conocer gente muy valiosa.

Por mi raza de bronce hablará el Espíritu Noble.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Durante el transcurso de los años los sistemas encargados de almacenar los datos para ser consultados y modificados han evolucionado. Desde archivos de texto hasta sistemas más complejos como los son los sistemas de bases de datos (SBD) actuales. Los SBD actuales tienen una alta confiabilidad y un tiempo de acceso eficiente en operaciones transaccionales. Los SBD que se ejecutan únicamente en una computadora, también conocidos como sistemas de bases de datos centralizados [11], pueden presentar restricciones tanto de hardware como de software que afectan su rendimiento. Ejemplos de estas restricciones pueden ser el espacio de almacenamiento, la cantidad de consultas concurrentes o la disponibilidad del sistema en caso de alguna falla en el equipo de cómputo donde se ejecuta. El rendimiento de un SBD centralizado también se ve afectado directamente al almacenar y/o consultar grandes volúmenes de datos así como tener un número de accesos elevado en un tiempo corto. Los SBD pueden tener dos enfoques de uso, los orientados a procesos de transacciones en línea[1] y los sistemas orientados al análisis de datos en línea también conocidos como OLAP[2]. Los sistemas OLAP, por lo general, utilizan una gran base de datos también llamada bodega de datos[3]. Esta bodega de datos está orientada a la consulta y al análisis de los datos para hallar patrones o información no trivial. Los avances tecnológicos en almacenamiento de datos y el desarrollo de las redes de computadoras han permitido desarrollar los sistemas de cómputo distribuido y en particular el desarrollo de los sistemas de bases de datos distribuidas (SBDD) con los cuales se busca aumentar el almacenamiento, hacer de manera más eficiente la consulta de grandes volúmenes de datos así como también mejorar la disponibilidad del sistema.

Algunos problemas presentes en un SBDD son principalmente problemas de comunicación o coordinación, ya que pueden dar lugar a inconsistencias o pérdida de datos. Para un SBDD es importante evitar y (en caso de existir) solucionar estos problemas para así tener un mejor control sobre los datos. Entre los problemas comunes pueden mencionarse el control de redundancia de datos o la correcta propagación de actualizaciones. Una base de datos distribuida (BDD) puede ser construida a partir de diferentes esquemas de distribución y sincronización de los datos. Dependiendo de los esquemas utilizados se puede obtener un funcionamiento correcto y eficiente de una BDD, aunque también la elección de un esquema inadecuado puede generar que el sistema falle o sea ineficiente. El presente trabajo se enfoca en el esquema de

[1] Su nombre en inglés Online Transaction Processing (OLTP)
[2] Su nombre en inglés Online Analitical Processing
[3] Su nombre en inglés Data Warehouse

replicación maestro-esclavo asíncrona[9] de una BDD. Como se mostrará más adelante el principal problema en este esquema es un problema conocido como *Inconsistencia de réplicas*. Por otro lado, la transmisión de grandes volúmenes de datos mediante los canales de comunicación actuales presentan dos principales problemas: tiempo de transmisión y seguridad de los datos. El artículo [8] propone una técnica de sincronización matemática para reconciliar conjuntos de datos entre dos sitios[4] con una complejidad de comunicación relativamente baja en comparación a la cantidad total de los datos. A través de éste y en base a otros trabajos relacionados se estudiará, implementará y experimentará la forma en la que la técnica de reconciliación de conjuntos[8] puede ser utilizada para detectar y reparar errores en una BDD con un esquema de replicación maestro-esclavo asíncrono.

Una BDD que tiene un esquema de replicación maestro-esclavo asíncrona se compone principalmente de *réplicas* [5] en distintos sitios. El esquema de replicación Asíncrona, utilizado para la propagación de actualizaciones, puede provocar que en determinado momento las réplicas se encuentren en un estado inconsistente [6]. Medir el grado de error presente en la BDD puede ser determinante en la interpretación del resultado de una consulta ya que los datos consultados pueden no estar actualizados. En el caso de detectarse una cantidad considerable de errores podría ser recomendable reparar la BDD antes de realizar la consulta. Este trabajo asume que el número de diferencias entre dos réplicas de una BDD es mucho menor comparado con el tamaño de datos almacenados en cada réplica, como puede ser el caso de bodegas de datos activas. Dada esta observación es necesario tener algoritmos eficientes para la detección de errores así como su reparación. De forma general en el esquema de replicación maestro-esclavo asíncrona cada réplica almacena un conjunto de datos, el objetivo del esquema es mantener las réplicas consistentes después de determinado tiempo o tras un cierto número de actualizaciones.

La reconciliación de dos conjuntos[7] puede llevarse a cabo mediante la obtención de la diferencia simétrica de los conjuntos. La diferencia simétrica de dos conjuntos que se encuentran en dos sitios distintos se puede hallar con operaciones de unión, intersección y complemento. Esta solución implica en la práctica una transmisión total de datos de un equipo de cómputo hacia otro y posteriormente operar ambos conjuntos para obtener la diferencia simétrica. Si los conjuntos de datos están situados en lugares geográficos distantes la comunicación representa un alto costo que aumenta de acuerdo al tamaño de los conjuntos sumado al costo de las operaciones realizadas para encontrar la diferencia simétrica. Esta forma de encontrar la diferencia simétrica es efectiva pero puede ser ineficiente tanto en tiempo de comunicación como en tiempo de cómputo si los conjuntos son grandes. Como caso particular de esta solución esta la actualización total de los conjuntos[8], es decir, la copia de un conjunto principal o actualizado de los datos en cada una de las réplicas. Debido a que esta operación puede ser costosa en este trabajo se estudiará una forma de llevar a cabo la detección y **reparación** de datos en una BDD de manera eficiente utilizando una técnica de reconciliación de conjuntos.

Como marco teórico se hace una introducción sobre BDD, en particular se enfatiza sobre el esquema de replicación maestro-esclavo asíncrono. También se explicará cómo funciona la técnica de *reconciliación de conjuntos*[8] así como su uso en algoritmos de medición de errores

[4] La definición de sitio se dará más adelante, por ahora un sitio se entenderá como un simple equipo de cómputo.

[5] Copias de los datos

[6] Que no contengan los mismos datos

[7] Los cuales pueden esta en sitios distintos

[8] Su término en inglés *Refresh*

en réplicas[6]. Estos algoritmos servirán como base para implementar la reparación de los errores detectados en la BDD.

La aportación de esta tesis se relaciona con el trabajo realizado en [6] sobre la implementación y experimentación del cálculo de métricas en una BDD que tiene un esquema de replicación maestro-esclavo asíncrona. El aporte de la tesis de maestría [2] es utilizado como referencia de experimentación para continuar la línea de trabajo del artículo [6]. Los experimentos realizados en [2] fueron diseñados para evaluar y comparar distintos algoritmos eficientes que calculan una métrica de inconsistencias de réplicas. En ellos se utiliza la clave primaria de las tablas para identificar cuando dos o más tuplas son distintas. Este trabajo no presenta los mismos experimentos ni el mismo código utilizado en [2], en su lugar se expone el desarrollo de una implementación diferente para abordar un problema distinto como lo es la reparación de los errores. En [6] se mencionan ideas de cómo se puede proceder a reparar una réplica una vez que se obtuvo una métrica sobre los errores. En [6] la métrica de inconsistencias detecta errores en cualquier atributo de la tupla, es decir, errores en atributos que no pertenecen a la clave primaria. En el artículo mencionado se proponen ideas sobre la reparación pero no se muestra un trabajo de experimentación, el cual es una parte sustancial de esta tesis.

De esta manera los capítulos iniciales expondrán conceptos y definiciones sobre bases de datos distribuidas, posteriormente se presentará la técnica de reconciliación de conjuntos utilizada y su aplicación práctica en una BDD. Como parte principal se describe la implementación y experimentación acerca de la reparación total[9] de una BDD con el esquema de replicación maestro-esclavo asíncrona utilizando la técnica de reconciliación de conjuntos que permite tener una complejidad de comunicación relativamente[10] baja. Mostrando así un contenido distinto al de [2] y novedoso respecto a la componente experimental de [6]. Como conclusión del trabajo se comentará el desarrollo del algoritmo propuesto y los resultados de la experimentación explicando ventajas y desventajas de la implementación realizada.

1.1. Objetivos

Anteriormente se mencionó que las cuestiones presentes al utilizar una BDD están ligadas al esquema y al comportamiento de la BDD. Para el esquema utilizado en este trabajo es de gran interés conocer el estado de cada una de las réplicas y posteriormente, en caso de presentar errores repararlos. Se considera también importante el desarrollo de herramientas eficientes y confiables para resolver estos problemas. El objetivo central de esta tesis consiste en implementar una solución al problema de reparación de una BDD que sigue un esquema de replicación maestro-esclavo asíncrona. A continuación se mencionan los objetivos que, una vez alcanzados, permitan llegar al objetivo principal:

Objetivos.

1. Estudiar el esquema de distribución y sincronización en una BDD maestro-esclavo asíncrono.
2. Comprender en qué situaciones pueden ocurrir problemas de inconsistencia de réplicas en una BDD que sigue un esquema maestro-esclavo asíncrono.

[9] Corrigiendo errores detectados en cualquier atributo y no solo en la clave primaria
[10] Relativa comparada con la cantidad total de los datos

3. Analizar el problema de reconciliación de conjuntos propuesto en [8] y la técnica que permite resolver este problema con una baja complejidad de comunicación.
4. Estudiar el algoritmo para medir inconsistencias de réplicas[6] que está basado en la técnica de reconciliación de conjuntos anterior.
5. Modificar el algoritmo de medición de inconsistencias para obtener un algoritmo capaz de reparar los errores detectados.
6. Estudiar las cuestiones necesarias para implementar el algoritmo en un SMBD.
7. Generar un diseño sobre la programación del algoritmo y llevarla a cabo.
8. Experimentar con la reparación implementada.
9. Establecer ventajas y desventajas de la reparación expuesta anteriormente.
10. Investigar acerca de trabajos relacionados con esta tesis.

1.2. Trabajos relacionados.

Como se mencionó esta tesis continúa la línea de investigación del artículo [6]. La importancia del artículo fue el mostrar cómo se puede utilizar la técnica de reconciliación de conjuntos del artículo [8] en un contexto de bases de datos distribuidas. Aún cuando el uso de la técnica sirvió para obtener métricas sobre la base de datos replicada no se utilizó para el fin principal de la reconciliación de conjuntos. Es por ello que la contribución principal de la presente investigación era atender esta cuestión.

Otro trabajo relacionado fuertemente es el artículo [8] donde se explica el funcionamiento de la técnica de reconciliación ya que el estudio del funcionamiento de ésta permitió conocer mejor las ventajas y limitaciones de la técnica utilizada.

A su vez también se considera importante el trabajo [5] en el cual se implementa la técnica de reconciliación de conjuntos. Esta aportación es el núcleo funcional de lo realizado en esta tesis.

En cuanto a la parte experimental hay dos trabajos importantes, el ya mencionado artículo [6] y la tesis de maestría [2]. En ambos se enfatizó en la experimentación para obtener y comparar algoritmos de obtención de métricas. El código utilizado y los planes de pruebas de estos trabajos sirvieron como base para implementar los algoritmos descritos a lo largo de esta tesis.

1.3. Motivación

La motivación inicial fue la experimentación realizada para el artículo [6]. En esta experimentación se aplicaron varios conocimientos teóricos y prácticos obtenidos durante la carrera de Ciencias de la Computación. Parte de esta motivación también fue la oportunidad de colaborar en una investigación científica, seria y de nivel internacional.

Después de participar en la investigación mencionada surgió una idea interesante enfocada en la aplicación de una solución de un problema general (reconciliación de conjuntos) a un caso

particular (base de datos replicadas). Esta cuestión permitió establecer varios objetivos de este trabajo enfocándose en probar que la solución general podía adaptarse al problema particular de reparación de una base de datos replicada.

Trabajar con una base de datos en un entorno distribuido también motivó de manera importante esta tesis, ya que actualmente los sistemas de bases de datos distribuidas son utilizados al tratar con grandes volúmenes de datos.

Durante la experimentación se observó que las primeras soluciones no aprovechaban al máximo los recursos de hardware disponibles, lo que llevó a experimentar en distintas arquitecturas, obteniendo resultados notables. El hecho de poder trabajar con distintas arquitecturas, obtener resultados comparables y que se pudieron adaptar a lo ya realizado fue otra motivación importante.

Por último, las motivaciones principal y final de este trabajo es experimentar sobre sistemas de bases de datos distribuidas y así contribuir de alguna forma al desarrollo de éstos.

1.4. Organización

La organización del trabajo es la siguiente:

En el **Capítulo 1** se presenta una introducción, trabajos relacionados y objetivos, así como la motivación de este trabajo.

El **Capítulo 2** trata de los sistemas de bases de datos distribuidas, se introducen conceptos y definiciones, en particular se estudia el esquema maestro-esclavo asíncrono y el problema de inconsistencia de réplicas que presenta dicho esquema.

En el **Capítulo 3** se expone y ejemplifica la técnica de **reconciliación de conjuntos**[8] utilizada, así como su adaptación para medir la inconsistencia de una BDD [6].

La propuesta de reparación se desarrolla en el **Capítulo 4**. El diseño y los detalles de implementación también son expuestos en este capítulo.

El textbfCapítulo 5 define el ambiente de desarrollo y de pruebas, un ejemplo de la ejecución de la reparación, experimentos y sus resultados así como una interpretación de estos últimos.

Por último el **Capítulo 6** presenta las conclusiones acerca del desarrollo del algoritmo, así como los resultados de la experimentación.

Capítulo 2

Sistemas de bases de datos distribuidas

Un Sistema de Bases de Datos Distribuidas (SBDD) almacena los datos de una base de datos en distintos equipos de cómputo. Cada una de las computadoras que forman parte del entorno distribuido es denotada como Sitio para enfatizar su distribución física. En cada sitio se tiene un Sistema Manejador de Bases de Datos (SMBD) para almacenar su porción de datos. La comunicación entre los sitios así como la transferencia de información se realiza a través de una red de computadoras. Por medio de la red se establece la comunicación entre los SMBD en diferentes sitios de manera que es también por este medio por el cual se puede acceder a los datos. Las ventajas principales contra un sistema centralizado son la disponibilidad, la autonomía y la distribución de los datos. Éste capítulo presenta conceptos, definiciones y esquemas en cuanto al almacenamiento de los datos y la coordinación entre los sitios de una Base de Datos Distribuida (BDD).

2.1. Conceptos y definiciones.

2.1.1. Sistema de bases de datos relacionales

Un dato es la representación simbólica, un atributo o característica de una entidad. Un dato por sí mismo no tiene valor semántico, pero en conjunto o mediante su procesamiento pueden proporcionar información. Una base de datos es un conjunto de datos interrelacionados y organizados bajo un mismo contexto almacenados en memoria secundaria de un equipo de cómputo para su consulta posterior. La descripción de los datos, sus relaciones, su semántica y las restricciones de los datos se encuentran definidas en lo que se denomina un modelo de datos. El modelo de datos utilizado en muchos Sistemas de Bases de Datos (SBD) y el utilizado por el SBD al que se enfoca este trabajo es el Modelo Relacional[4].

El Modelo Relacional fue propuesto por Edgar Frank Codd en 1970. Dicho modelo contempla el concepto de **relación** como un conjunto de tuplas [1]. La manipulación de las relaciones se hace mediante un lenguaje formal que describe la forma de hacer una consulta, este lenguaje

[1] Secuencia finita y ordenada de datos

es el Álgebra Relacional. En adelante al referirse a una base de datos se asumirá que ésta tiene un modelo relacional. Las relaciones se pueden representar a través de una tabla en la que cada fila representa una tupla, y cada dato de la tupla forma parte de una columna. Un sistema de bases de datos por lo general cuenta con una arquitectura de 3 capas (2.1). La más baja es la capa de almacenamiento donde se define como se guardaran y recuperaran los datos a nivel de disco. La capa intermedia o lógica define la forma de estructurar, relacionar y acceder a los datos. Por último la capa más alta es una capa de vista en la que el usuario del sistema accede a los datos a través de programas o aplicaciones.

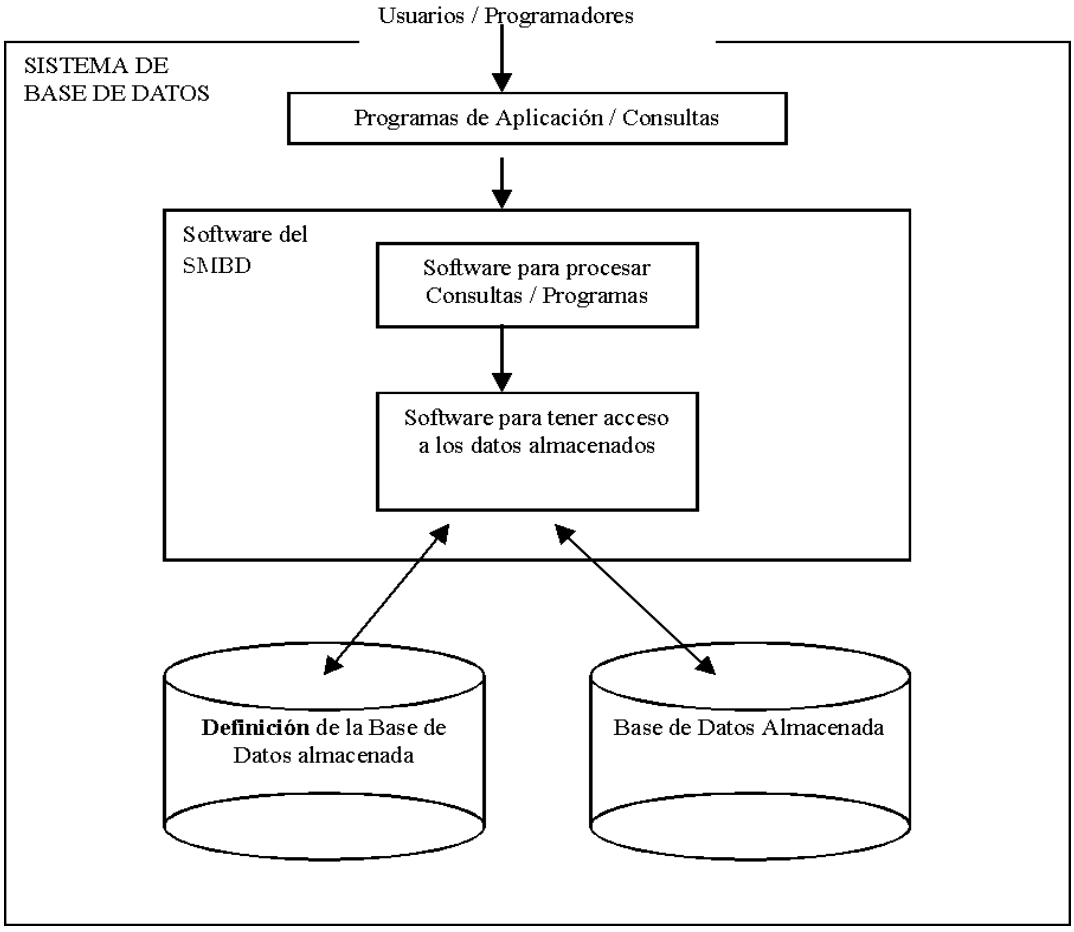

Figura 2.1: Arquitectura de tres capas de un Sistema de Base de Datos

Un Sistema Manejador de Bases de Datos Relacionales (simplemente SMBD) es software especializado en manejar los datos en un modelo relacional y cuyo lenguaje de definición, consulta y manipulación utilizado es el SQL (Structured Query Language). El SMBD actúa como interfaz entre los usuarios del sistema y el modelo de datos. Un usuario es una entidad o grupo de entidades que requieren algún servicio[2] proporcionado por el SMBD, dicha entidad puede ser una persona u otro sistema, en particular otro SMBD.

[2]Ingresar, obtener o modificar datos principalmente

2.1.2. Sistema Distribuido

Un sistema distribuido es un sistema cuyos componentes tanto de software como de hardware están en sitios diferentes, los cuales se comunican y coordinan mediante paso de mensajes para lograr un objetivo común. Como componentes importantes se pueden mencionar:

- Los sitios o nodos, que son equipos de cómputo que ofrecen algún recurso como almacenamiento, memoria o tiempo de procesamiento a otros sitios.
- La red es el medio de comunicación entre los sitios del sistema distribuido.
- Software encargado de la coordinación y paso de mensajes entre los sitios.
- Software que realiza una tarea de forma distribuida aprovechando los recursos de los sitios que componen el sistema distribuido.

Algunas de las ventajas que presenta un sistema distribuido son las siguientes:

- Concurrencia: Ejecución de varios procesos al mismo tiempo, multiprocesamiento.
- Funcionalidad: Cada sitio puede ofrecer recursos específicos o ejecutar tareas distintas a las de otro sitio.
- Distribución del trabajo: Cada nodo ejecuta una o varias partes de un trabajo común al mismo tiempo.
- Distribución física: El sistema puede estar disperso geográficamente.

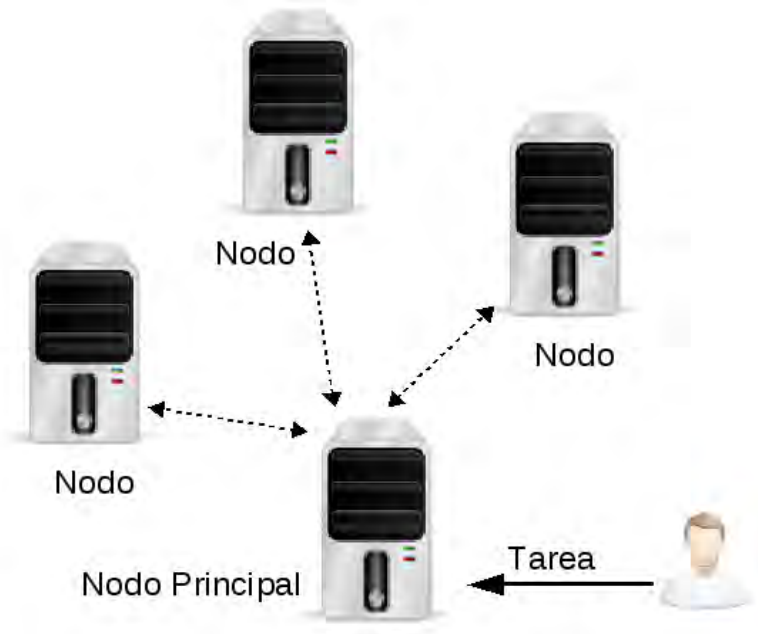

Figura 2.2: Estructura básica de un Sistema Distribuido

Un componente que influye en gran medida en el correcto y eficiente funcionamiento de un sistema distribuido es la red de comunicación. Una red puede presentar daños físicos por desastres naturales, accidentes o deterioro de sus componentes. Por otro lado la red puede exponer los datos provocando riesgos de seguridad donde personas no autorizadas pueden acceder a ellos. Otro problema puede ser el tráfico provocado por un alto flujo de datos el cual puede evitar que éstos lleguen a su destino. En términos de redes los paquetes de datos son la unidad fundamental de transporte de la información. Los problemas de una red se pueden catalogar como problemas de:

- Tráfico de la red.
  Influye en el tiempo de espera de los paquetes de datos.
- Costo de la transmisión.
  El número de paquetes de datos y la cantidad de datos en cada paquete tienen un costo de transmisión.
- Fiabilidad de la red.
  Probabilidad de que la red funcione correctamente, es decir, que todos los paquetes lleguen a su destino y en un tiempo determinado.
- Seguridad: Sólo los usuarios autorizados pueden recibir los paquetes y acceder a los datos enviados por la red.

Como se puede observar, los sistemas distribuidos presentan características interesantes y funcionalidades de gran ayuda. De la misma forma presentan problemas inherentes a sus componentes que, dependiendo del problema, pueden ser evitados o controlados para evitar que influyan en el desarrollo de una tarea común.

2.1.3. Sistema de bases de batos distribuidas

Un sistema de bases de datos distribuidas (SBDD) almacena la base de datos en distintas computadoras o sitios. Al tratarse de un sistema distribuido se debe considerar la red de comunicación a través de la cual los sitios podrán coordinarse o intercambiar datos para realizar un trabajo común. Por lo general cada sitio mantiene un SMBD centralizado que al trabajar de manera conjunta puede proporcionar ventajas como las siguientes:

- Concurrencia: En cada sitio pueden procesarse distintas consultas a la base de datos, la participación de cada sitio para dar un resultado a la consulta depende de la distribución de los datos.
- Funcionalidad: Sitios dedicados a recibir información y sitios únicamente de consulta.
- Distribución del trabajo: Consultas complejas podrían optimizarse si cada sitio ejecuta una parte de la consulta, el problema consiste en dividir el problema, distribuirlo y combinar los resultados devueltos por cada sitio.
- Físicas: Al estar dispersa geográficamente se puede mejorar el tiempo de acceso a los datos. También puede que cada sitio almacene sólo una porción de los datos en lugar de almacenar y mantener una copia completa de toda la base de datos, es decir, distribuir el almacenamiento de los datos.

Un SBDD puede aprovechar distintas ventajas de un esquema distribuido dependiendo del enfoque o comportamiento requerido por los usuarios. Entre las razones para utilizar un SBDD destacan tres:

- Datos compartidos.
  La principal ventaja es poder disponer de un entorno donde los usuarios puedan acceder desde distintas ubicaciones a los datos que residen en otro lugar.

- Autonomía.
  Como resultado de la distribución y compartición de los datos cada sitio es capaz de mantener un grado de control sobre los datos que almacena localmente. Un administrador global de un SBDD delega a administradores (en cada sitio) responsabilidades que permiten tener diferentes comportamientos. Esta autonomía local es considerada una de las grandes ventajas de los SBDD.

- Disponibilidad.
  Los fallos en un sitio pueden no significar un problema en todo el sistema, ya que el resto de los sitios pueden seguir funcionando. Por otro lado también puede funcionar como un mecanismo de tolerancia a fallas, es decir, si el sistema detecta errores en un sitio, puede llevar a cabo acciones que permitan reparar ese sitio. La capacidad de continuar funcionando a pesar de un fallo en uno de los sitios mantiene la disponibilidad.

Los datos al ser la unidad básica de un SBD tienen que ser tratados de manera cuidadosa. Los SMBD así como las redes de datos han desarrollado mecanismos para mantener los datos seguros, tales como el manejo de usuarios y permisos sobre los datos o métodos de encriptación. Por otra parte un SMBD está resguardado también por el sistema operativo aumentando así la seguridad de los datos por los mecanismos que ofrece el propio sistema operativo. La red al ser un medio externo a los equipos de cómputo presenta más riesgos de seguridad para los datos que transmite. Al crear un SBDD se tienen que contemplar los riesgos que conlleva la transmisión de los datos en una red. El SBDD trata de evitar principalmente los problemas de tráfico y costo de transmisión de los datos, proponiendo esquemas de distribución que buscan reducir la cantidad de datos transmitidos. Un SBDD puede ser catalogado como homogéneo o heterogéneo. Heterogéneo si los sitios utilizan esquemas lógicos de los datos o SMBD diferentes. Un SBDD heterogéneo puede proporcionar ventajas limitadas para la cooperación en el procesamiento de algunas tareas. En cambio, para un SBDD homogéneo todos los sitios emplean un mismo SMBD o un mismo esquema lógico, permitiendo una mejor cooperación para atender las peticiones de manera conjunta. Como es de esperarse hay distintas variables a considerar cuando se opta por utilizar un SBDD. Una de las variables que más influye en el esquema es el almacenamiento de los datos. De acuerdo al modelo relacional, una relación puede almacenarse de acuerdo a 2 enfoques en una base de datos distribuida:

- Fragmentación.

- Replicación.

Estos esquemas se explican en las secciones siguientes del capítulo.

2.2. Fragmentación de los datos

Este almacenamiento distribuido de una relación r fragmenta o divide en distintos fragmentos $r_1, r_2, ..., r_n$ la relación r. Los fragmentos contienen datos que permiten reconstruir la relación original. Uno o varios fragmentos pueden ser almacenados en distintos sitios. Hay principalmente 2 formas de dividir una relación:

- Fragmentación vertical.
- Fragmentación horizontal.

2.2.1. Fragmentación vertical

Una relación es un conjunto de tuplas y cada tupla es una lista de datos finitos y ordenados. Por ello una tabla puede ser la representación de una relación. Fragmentar verticalmente una relación consiste en elegir un subconjunto de columnas de la tabla que representa la relación. Siguiendo el lenguaje formal de consulta de la relaciones, el álgebra relacional, la fragmentación vertical puede verse como la proyección de la relación r de acuerdo a un conjunto de atributos.

Sea r una relación y $a_1, a_2, ..., a_n$ subconjuntos de atributos de r, es decir:

$$atributos\, de\, r = a_1 \cup a_2 \cup ... \cup a_n$$

entonces cada fragmento r_i se define como:

$$r_i = \Pi_{a_1}(r)$$

La reconstrucción de r debe realizarse mediante la reunión de los fragmentos r_i

$$r = r_1 \bowtie r_2 \bowtie ... \bowtie r_n$$

Para asegurar que la reconstrucción de la relación r sea correcta y completa, cada fragmento r_i deberá contener la clave primaria de manera que a través de la operación de reunión de fragmentos sobre la clave primaria se obtenga la relación r original.

2.2.2. Fragmentación horizontal.

En este esquema la fragmentación es sobre conjuntos de tuplas completas, es decir, sobre conjuntos de filas de la tabla que representa a la relación. La operación en álgebra relacional que nos permitiría hacer esta fragmentación es la selección. De tal forma que cada fragmento de la relación r es una selección que utiliza un predicado p_i :

$$r_i = \sigma_{p_i}(r)$$

Para reconstruir r se realiza una unión de todos los fragmentos horizontales r_i

$$r = r_1 \cup r_2 \cup ... \cup r_n$$

Es importante recalcar que al tratarse de tuplas completas, lo que se debe asegurar es que los predicados p_i utilizados en la selección cubran todos los valores existentes de la clave primaria, ya que de no hacerlo la reconstrucción de la relación r no sería completa. Para la BDD que se usará más adelante no se considera una fragmentación en el esquema de almacenamiento de los datos pero se retomaran ideas de fragmentación para poder manejar los datos de manera más eficiente.

2.3. Replicación de los datos

En este enfoque el SBDD mantiene copias de los datos en distintos Sitios. La replicación puede ser parcial o total, en uno, muchos o en todos los sitios[3]. Las características principales de la replicación de los datos son entre otras:

- Disponibilidad. Entre más completa sea una réplica la disponibilidad aumenta.
- Eficiencia en accesos de lecturas. Si la lectura involucra datos que se encuentran en distintos sitios el tiempo de comunicación aumenta el tiempo de lectura, en cambio, cuantos más datos replicados existan es mayor la probabilidad de encontrar los datos necesarios en un sitio.
- Tiempo de actualizaciones. La actualización a un dato con varias réplicas tiene que propagarse en cada una de ellas, aumentando el tiempo de actualización o provocando también interbloqueos. Esta propagación debe ser correcta o de lo contrario una o varias réplicas quedarían en un estado inconsistente.

Ya que la distribución de los datos afecta el comportamiento de un SBDD, podemos clasificar un esquema de una BDD de acuerdo al enfoque de replicación de los datos.

2.3.1. Esquema maestro-esclavo

Un sitio maestro es un sitio que permite tanto accesos de lectura como de escritura por parte de los usuarios de un SBDD. Un usuario del SBDD es cualquier entidad fuera del SBDD que requiera hacer una consulta, inserción o actualización de datos en el SBDD.

Por otro lado un sitio esclavo sólo recibe consultas, inserciones o actualizaciones como peticiones de un único usuario, el sitio maestro. El sitio esclavo también puede permitir peticiones de lectura por parte de otros usuarios del SBDD aparte del sitio maestro.

Este esquema mantiene la disponibilidad y la eficiencia en lecturas pero puede presentar problemas de tiempo de actualización ya que las actualizaciones solo se pueden hacer en el sitio maestro y posteriormente deben propagarse a los sitios esclavos lo cual implica tiempo de comunicación. La eficiencia en lecturas solamente mejora si los usuarios del SBDD tienen acceso de lectura a los esclavos y no solo al maestro. Otra ventaja es la tolerancia a fallos ya que si el sitio maestro llega a fallar en determinado momento, un sitio Esclavo puede convertirse en sitio maestro y continuar el funcionamiento del sistema.

[3] La replicación de un conjunto de datos en todas las réplicas se conoce como *Replicación Completa*

La réplica de los datos puede ser parcial o total. Cuando se considera una replicación total de los datos se referirá al conjunto original de los datos como **copia maestra** y a las copias de los datos en otros sitios se les conocerá como **réplica**. Mantener los datos de la copia maestra en todas las réplicas implica un problema de propagación de actualizaciones o sincronización de los datos. La Copia Maestra de los datos puede ser almacenada en cualquier réplica, por cuestiones prácticas y de notación se considerará siempre que la copia maestra estará almacenada en el sitio maestro y de esta manera las réplicas de los datos serán almacenadas en los sitios esclavos.

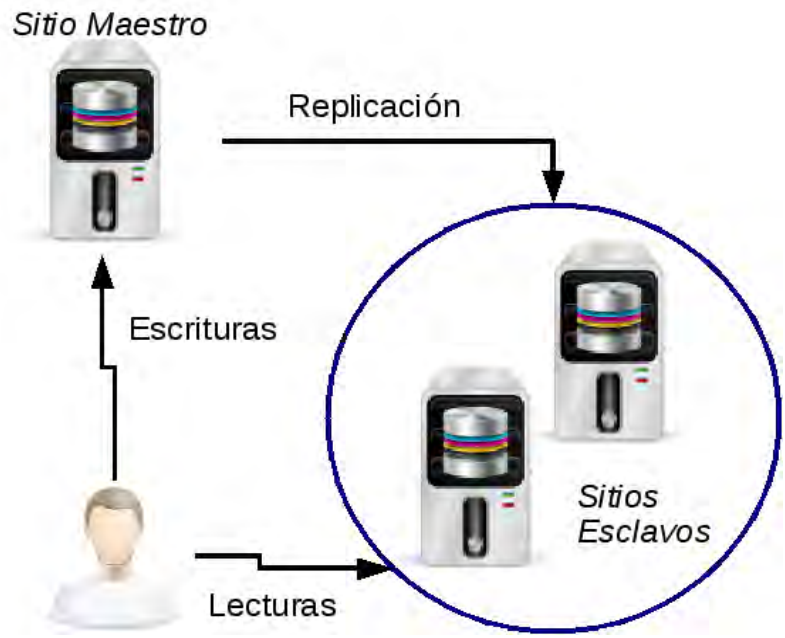

Figura 2.3: Replicación de una base de datos con el esquema maestro-esclavo

2.3.2. Esquema multimaestro

Cada sitio en este esquema es un maestro y por lo tanto cada sitio puede recibir peticiones de lectura y escritura por parte de los usuarios de la BDD. Si la replicación de los datos es alta, la administración de este esquema se vuelve complicada. Mantener varios sitios con los mismos datos provoca que el desempeño del sistema disminuya. Por otro, lado la consulta de los datos puede mantenerse eficiente ya que la probabilidad de que un sitio tenga los datos requeridos es alta, aunque la propagación correcta de las actualizaciones incrementa el número de conflictos en la administración del sistema.

Una cuestión importante que presenta este esquema es un problema de interbloqueos ya que la actualización de un mismo dato en varios sitios implica una condición de competencia para determinar cuál de las actualizaciones es la que debe llevarse a cabo o en que orden. Aunque el problema de interbloqueos puede ser tratado mediante el correcto uso de transacciones y bitácoras, el tiempo que tarda una consulta que bloquea un dato para su actualización provoca que los sitios que desean actualizar el mismo dato también deban esperar.

2.3.3. Sincronización de bases de datos distribuidas

Una BDD que no utiliza un esquema de replicación provoca que cada sitio almacene su propia porción de datos, actuando como un sistema centralizado para administrar un conjunto específico de datos. Por el contrario, un esquema de replicación en una BDD obliga a mantener las réplicas de los datos actualizadas respecto a una copia maestra. En este caso el tiempo que transcurre entre las actualizaciones a la copia maestra y la propagación de la actualización en las réplicas da lugar al problema de *inconsistencia de réplicas*.

En el contexto de las BDD se define la sincronización como el proceso que deben realizar conjuntamente los sitios para que cada una de las réplicas contenga los mismos datos que la copia central. El tiempo es un factor importante en este proceso, ya que en el momento de la sincronización la BDD puede bloquearse completamente para evitar errores. Otro factor importante es el esquema de almacenamiento, como lo son los esquemas de replicación maestro-esclavo y multimaestro. Para este trabajo solamente se estudia el problema de la Sincronización en una BDD que sigue un esquema maestro-esclavo.

Como se mencionó en la sección referente al esquema de almacenamiento maestro-esclavo, existe un conjunto de datos principal, una *copia maestra* de los datos, la cual se almacena en un sitio particular, el sitio maestro. Por su parte cada esclavo almacenará una réplica de la copia maestra.

Una vez que se define dónde se encuentran los datos a sincronizar es necesario determinar en qué momento debe realizarse la sincronización. Pueden existir distintas condiciones para llevar a cabo este proceso, por ejemplo: programar la sincronización en periodos de tiempo, al darse un determinado número de modificaciones en la copia maestra o simplemente cuando el administrador del sistema lo considere necesario. Estas condiciones básicamente se refieren a un cierto estado de las replicas respecto a la copia maestra. El comportamiento de la BDD distribuida puede requerir, o no, que la inconsistencia de replicas sea nula en todo momento.

- **Síncronamente**: Llevar a cabo el proceso de Sincronización cada vez que ocurra una actualización a la copia maestra, con lo cual el tiempo para realizar la actualización será igual al tiempo que tarda en ser propagada la actualización a todas las réplicas de ese dato. En este esquema la inconsistencia en las réplicas es nula después de la ejecución de cualquier petición de borrado, inserción o actualización de datos.
- **Asíncronamente**: En éste caso la BDD permite un grado de inconsistencias, al llegar a este límite de inconsistencias la sincronización de las réplicas debe llevarse a cabo y dejar un grado nulo de inconsistencias. Las actualizaciones antes de llegar al punto de realizar la sincronización son ejecutadas eficientemente ya que solamente se efectúan en la copia maestra, por lo que el proceso de sincronización debe ser lo más eficiente posible.

Respecto al modo asíncrono se puede mantener una bitácora en la copia maestra con las últimas actualizaciones que después se ejecutarán en las réplicas con el fin de sincronizarlas. Esta solución aunque puede ser eficiente depende de la fiabilidad de la bitácora y no de los datos, tras una pérdida de la bitácora, la misma pudiera no ser recuperada y por lo tanto no efectuar la sincronización de manera eficiente. El uso de bitácoras también puede verse limitado si se tratan de integrar bases de datos de distintas fuentes en las cuales no se utiliza una bitácora. En este trabajo se considerara la sincronización como una reparación de errores de inconsistencia,

ya que se asume que los errores están presentes en la BDD y se desea eliminarlos. La reparación que se expondrá a lo largo del trabajo pretende ser eficiente y autocontenida.

Los capítulos posteriores de este trabajo se encargan de exponer y explicar una forma de llevar a cabo una sincronización o reparación atendiendo cuestiones prácticas como lo son el almacenamiento y la transmisión de los datos. Para ello se utilizará una técnica de sincronización[4] con una complejidad de comunicación baja adaptada a una base de datos replicada con un esquema maestro-esclavo asíncrono.

[4] Utilizando los términos de reconciliación y reparación en lugar de sincronización

Capítulo 3

Técnica de reconciliación de conjuntos.

De acuerdo al esquema de distribución de los datos en una Base de Datos Distribuida (BDD) una cuestión inherente es cómo mantener en cada sitio los mismos datos, si así se desea, y cuál es la mejor forma de llevar a cabo esta operación. En la teoría hay distintos protocolos o algoritmos que permiten realizar dicha tarea si se considera a cada sitio como un conjunto de datos. De esta manera el problema a tratar en este trabajo se presenta con conjuntos de datos que se hallan en sitios [1] geográficamente distribuidos. A continuación se estudia una técnica que se encarga de resolver el problema de reconciliación de conjuntos con una complejidad de comunicación muy baja y ligada al número de inconsistencias existentes entre las réplicas, tal técnica se presenta formalmente en el artículo [8]. Como puede deducirse por su nombre, la reconciliación de conjuntos se encarga de establecer una igualdad entre conjuntos, de manera formal se trabajará con la definición [2] hecha en el artículo [8].

Gran parte de este capítulo tiene como base los artículos [8, 6] y debido a su importancia en el trabajo a desarrollar se considera exponerlos para explicar los conceptos e ideas que se plantean en los capítulos posteriores.

3.1. Reconciliación de conjuntos

Dado un par de sitios S_A y S_B, cada uno con un conjunto de datos A y B respectivamente, se debe determinar la unión de los dos conjuntos con una cantidad mínima de comunicación respecto al número de intercambios entre los sitios y al número de datos intercambiados, se referirá a este problema como *Problema de reconciliación de conjuntos*. [3]

Como puede observarse en la definición, el problema involucra sólo dos sitios pero puede involucrar más sitios si se establece un protocolo u organización de interacción entre los sitios. También se enfatiza la necesidad de disminuir lo más posible la complejidad de comunicación, para ello la técnica presentada a continuación esta basada en una representación particular

[1] Hosts o Equipos de cómputo.
[2] Definición orientada a sistemas distribuidos.
[3] Su nombre en inglés: *Set Reconciliation Problem*

de los datos. La representación de los datos es necesaria para aplicar un método numérico cuyo resultado se utilizará para para establecer un algoritmo que dé solución al problema de reconciliación de conjuntos.

De acuerdo a la definición[4], sean los sitios S_A y S_B cuyas diferencias son $\Delta_A = A - B$ y $\Delta_B = B - A$ de tamaño m_A y m_B respectivamente. Sea también $m = m_A + m_B$ el número total de diferencias entre ambos sitios. El cálculo de estas diferencias, conocida también como **diferencia simétrica**, es la clave de la técnica de reconciliación de conjuntos. Una vez calculadas las diferencias, en cada sitio se pueden realizar operaciones que determinen los datos inconsistentes. Estos resultados pueden ser aprovechados por distintos algoritmos, ya sea para calcular métricas de los datos o para reparar los conjuntos.

3.1.1. Polinomio característico

La representación de los datos se hace mediante el cálculo de un *Polinomio característico* o simplemente *polinomio* [7, 3]. Un polinomio característico es, de cierta forma, un resumen particular del conjunto de datos, ya que para construirlo se consideran todos los datos contenidos en el conjunto.

El cómputo que utilizará los polinomios para hallar las diferencias entre conjuntos de datos es un cálculo numérico. Por lo tanto es necesaria una representación numérica de los datos, esta puede ser una representación de cadenas de bits o cualquier otra representación numérica. De manera general se define un polinomio característico $\varphi_A(Z)$ de un conjunto $A = \{x_1, x_2, ..., x_n\}$ como :

$$\begin{aligned} \varphi_A(Z) &= (Z - x_1)(Z - x_2)...(Z - x_n) \\ &= Z^n - \sigma_1(A)Z^{n-1} + ... + (-1)^n\sigma_n(A) \end{aligned} \tag{3.1}$$

Cada coeficiente $\sigma_i(A)$ es conocido como un polinomio simétrico elemental y representa cada uno de los coeficientes del polinomio característico. Por otra parte se puede observar que cada elemento del conjunto A representa una raíz del polinomio. Como se mencionó, los elementos de A son los datos en una representación numérica, la cual debe estar dentro de un campo numérico $\mathcal{F}$ o ser mapeada a valores en dicho campo para tener una cantidad acotada de números con los cuales hacer las operaciones. Por ello el campo $\mathcal{F}$ elegido deberá ser mayor al máximo elemento de A. De esta manera, las operaciones realizadas con los polinomios deben realizarse en aritmética modular, con modulo $\mathcal{F}$. Z sera un conjunto de elementos en $\mathcal{F}$ llamados *puntos de evaluación*. Un punto de evaluación se define como un número $z \in \mathcal{F}$ tal que $z \notin A \cup B$.

3.1.1.1. Actualización del polinomio característico

La forma de calcular el polinomio involucra todos los datos del conjunto A por lo que su construcción tiene un orden de complejidad lineal $O(|Z| \cdot |A|)$, donde $|Z|$ sera un valor fijo donde cada elemento de A se evalúa en cada punto de evaluación en Z. De esta forma, al añadir un elemento al conjunto, el polinomio necesitaría ser actualizado. La idea de re-calcular todo el polinomio cuando sólo se ha añadido un elemento al conjunto puede ser ineficiente ya que la cantidad de datos en A es considerable.

[4] A lo largo de esta sección utilizamos la notación del artículo [8]

En la ecuación (3.1) al añadir un nuevo elemento x_{n+1} al conjunto A, la ecuación sólo se ve afectada al añadir el término $(Z - x_{n+1})$ en la multiplicación. De manera similar, se puede actualizar el polinomio característico en el caso de eliminar un elemento en A, ya que al quitar el elemento x_1 de A polinomio puede ser dividido por $(Z - x_1)$, es decir: $\frac{\varphi_A(Z)}{(Z-x_1)}$.

En ambas operaciones, actualizar un polinomio característico tiene un orden de complejidad $O(|Z|)$.

3.1.1.2. Funciones racionales y polinomios característicos

La idea de utilizar polinomios característicos de dos conjunto de datos para hallar la diferencia simétrica se explicara a continuación.

Sean S_A y S_B dos sitios con conjuntos de datos A y B respectivamente, los datos se encuentran dentro de un campo numérico $\mathcal{F}$, los datos en los sitios pueden ser expresados como:

$$A = (A \cap B) \cup (\Delta_A)$$
y
$$B = (A \cap B) \cup (\Delta_B)$$

Aplicando la función (3.1) a cada conjunto en los puntos de evaluación Z, se obtienen las ecuaciones siguientes:

$$\varphi_A(Z) = \varphi_{A\cap B}(Z) \cdot \varphi_{\Delta_A}(Z) \tag{3.2}$$

$$\varphi_B(Z) = \varphi_{A\cap B}(Z) \cdot \varphi_{\Delta_B}(Z) \tag{3.3}$$

La relación entre las ecuaciones (3.2) y (3.3) tienen como término común $\varphi_{S_A\cap S_B}(Z)$ el cual se cancela dejando la relación como una función racional.

$$\frac{\varphi_A(Z)}{\varphi_B(Z)} = \frac{\varphi_{\Delta_A}(Z)}{\varphi_{\Delta_B}(Z)} = f(Z) \tag{3.4}$$

Si bien los grados de los polinomios característicos pueden ser muy grandes, al eliminar los términos comunes, los grados de la funciona racional $f(Z)$ resultante son menores. También es importante denotar que los grados de los polinomios característicos son precisamente la cardinalidad de los conjuntos, es decir, m_A es el grado del polinomio $\varphi_A(Z)$ ya que se utilizaron todos los datos en A para su construcción, de igual forma el grado de $\varphi_B(Z)$ es m_B. Y dado que la función racional se formo a partir de los polinomios característicos entonces los grados del numerador y denominador de la función racional $f(Z)$ son, respectivamente, m_A y m_B. De esta manera y ya que en ambos conjuntos A y B el numero de elementos cancelados es el mismo, por estar en la intersección, se tiene la siguiente igualdad $|A|-|B| = |\Delta_A|-|\Delta_B| = m_A - m_B$.

El objetivo de obtener los valores de la diferencia simétrica a través de los polinomios característicos involucra resolver dos problemas numéricos principalmente:

- Interpolar una función racional a partir de los valores de $f(Z)$ y encontrar el polinomio numerador y denominador de esa función racional.
- Factorizar polinomios (numerador y denominador de la función racional interpolada) para obtener las raíces que corresponden a valores de la diferencia simétrica por separado.

3.1.2. Interpolación de una función racional y factorización

El cálculo de los polinomios para cada conjunto de datos se realiza tomando en cuenta una cota sobre el número de diferencias entre los conjuntos. Esta cota $\bar{m} > m = m_A + m_B$ permite definir el número de puntos de evaluación para el cálculo del polinomio. De tal manera que la evaluación de un polinomio característico en un conjunto Z de $\bar{m}$ puntos de evaluación para un conjunto $A = \{x_1, x_2, ..., x_n\}$ es:

$$\begin{aligned}\varphi_A(z_1) &= (z_1 - x_1)(z_1 - x_2)...(z_1 - x_n)\\ \varphi_A(z_2) &= (z_2 - x_1)(z_2 - x_2)...(z_2 - x_n)\\ &\vdots\\ \varphi_A(z_{\bar{m}}) &= (z_{\bar{m}} - x_1)(z_{\bar{m}} - x_2)...(z_{\bar{m}} - x_n)\end{aligned}$$

Dados dos sitios S_A y S_B de los cuales se obtienen sus polinomios característicos P y Q respectivamente al aplicar φ. La función racional $f(Z)$ para el conjunto Z de $\bar{m}$ puntos de evaluación se muestra en la siguiente ecuación:

$$f(Z) = \frac{\varphi_A(Z)}{\varphi_B(Z)} = \frac{P(Z)}{Q(Z)} \tag{3.5}$$

Dado que Z tiene varios elementos la ecuación (3.5) se expande como sigue:

$$\begin{aligned}f(z_1) &= \frac{(z_1-x_1)(z_1-x_2)...(z_1-x_i)\cancel{(z_1-x_{i+1})}}{\cancel{(z_1-x_{i+1})}...(z_1-x_n)} = \frac{P(z_1)}{Q(z_1)}\\ f(z_2) &= \frac{(z_2-x_1)(z_2-x_2)...(z_2-x_i)\cancel{(z_2-x_{i+1})}}{\cancel{(z_2-x_{i+1})}...(z_2-x_n)} = \frac{P(z_2)}{Q(z_2)}\\ &\vdots\\ f(z_{\bar{m}}) &= \frac{(z_{\bar{m}}-x_1)(z_{\bar{m}}-x_2)...(z_{\bar{m}}-x_i)\cancel{(z_{\bar{m}}-x_{i+1})}}{\cancel{(z_{\bar{m}}-x_{i+1})}...(z_{\bar{m}}-x_n)} = \frac{P(z_{\bar{m}})}{Q(z_{\bar{m}})}\end{aligned}$$

A continuación se en listan algunas definiciones y propiedades utilizadas para explicar cómo se realiza la interpolación y factorización de la función racional.

- Una función racional es mónica si los coeficientes principales de su numerador y denominador son iguales a 1.
- Sea una función racional definida como $f(Z) = P(Z)/Q(Z)$. Si el numerador y denominador son primos relativos se dice que f es reducida, es decir, no se puede simplificar.
- Dos funciones racionales P_1/Q_1 y P_2/Q_2 son equivalentes si $P_1Q_2 = Q_1P_2$.
- Un conjunto de soporte V es un conjunto de elementos $(z_i, f_i) \in \mathcal{F}^2$ con z_i distintos entre si. Una función $f(Z)$ satisface V si $f(z_i) = f_i$ para todo par $(z_i, f_i) \in V$.

Como se menciona en la sección anterior los datos, así como los puntos de evaluación, se encuentran en un campo numérico $\mathcal{F}$, dado que no tienen factores comunes en $\mathcal{F}$, $f(Z)$ se dice que es reducida. También se observa que $f(Z)$ es mónica, ya que el coeficiente principal en ambos polinomios P y Q es 1.

El problema a resolver se enuncia a continuación:
Encontrar una función racional $f(Z)$ mónica que satisfaga al conjunto de soporte V de tamaño $\bar{m}$ tal que la suma de los grados del numerador y denominador sea menor o igual a $\bar{m}$ y la diferencia entre los grados sea d.

Este problema involucra dos cuestiones principales, por un lado asegurar que la solución existe y es única; y por otro lado encontrar un algoritmo eficiente para reconstruir la función racional dado el conjunto de soporte V.

De manera particular se tiene que el conjunto de soporte V puede ser construido por los valores de $\frac{\varphi_A(Z)}{\varphi_B(Z)} = \frac{P(Z)}{Q(Z)}$, y a su vez se tiene que $|A| - |B| = m_A - m_B = d$.

Para comprobar la unicidad y existencia el artículo [8] presenta y demuestra el teorema siguiente:

Teorema 1 *Sea V un conjunto de soporte con $\bar{m}$ elementos sobre un campo $\mathcal{F}$. Suponemos que existen dos funciones racionales que satisfacen V y que la suma de sus grados no es mayor que $\bar{m}$.Entonces ambas funciones son equivalentes.*

De este teorema se deduce que si hay dos funciones racionales equivalentes entonces la interpolación racional[5] es única.

El procedimiento para encontrar la función equivalente $g(Z) = \frac{P'(Z)}{Q'(Z)}$ se muestra en la sección 3.2.2 del artículo [8], para ello se asume que existe una función racional reducida, $f(Z)$, con grados (m_A, m_B) tales que $m_A + m_B \leq \bar{m}$ y $m_A - m_B = d$, el conjunto de soporte V se define como (z_i, f_i) donde $f_i = f(z_i)$. Dado que el teorema necesita de cotas para los grados de los polinomios numerador y denominador, éstas se establecen de acuerdo a la cota sobre el máximo número de diferencias $\bar{m}$:

$$\begin{aligned} m_A &\leq \lfloor (\bar{m} + d)/2 \rfloor = \bar{m_A} \\ m_B &\leq \lfloor (\bar{m} - d)/2 \rfloor = \bar{m_B} \end{aligned}$$

La manera de definir las cotas permite aplicar el teorema (1) y así se asegura la unicidad de la función equivalente, donde los grados de P' y Q' corresponden a $\bar{m_A}$ y $\bar{m_B}$ respectivamente.

De acuerdo a la ecuación (3.1) en su forma de polinomios simétricos elementales se puede representar la función racional como:

$$f(Z) = \frac{p_0 Z^{m_A} + p_1 Z^{m_A - 1} + \ldots + p_{m_A}}{q_0 Z^{m_B} + q_1 Z^{m_B - 1} + \ldots + q_{m_B}} \tag{3.6}$$

Además es de notar que $\bar{m}_A - m_A = \bar{m}_B - m_B$, por lo cual basándose en la ecuación (3.6) se puede definir una función racional equivalente de la siguiente manera: $f(Z)$ multiplicada por $Z^{\bar{m_A} - m_A}$ en el numerador y denominador:

$$f(Z) = \frac{p_0 Z^{\bar{m_A}} + p_1 Z^{\bar{m_A} - 1} + \ldots + p_{\bar{m_A}}}{q_0 Z^{\bar{m_B}} + q_1 Z^{\bar{m_B} - 1} + \ldots + q_{\bar{m_B}}} \tag{3.7}$$

[5] Basada en interpolar los coeficientes de los polinomios

Dado que $f(Z) = \frac{P(Z)}{Q(Z)}$ y además $\frac{P(Z)}{Q(Z)} = \frac{P'(Z)}{Q'(Z)}$ se puede obtener la relación $P'(z_i) = f_i Q'(z_i)$ la cual se puede expandir a:

$$z_i{}^{\bar{m}_A} + p_1 z_i{}^{\bar{m}_A - 1} + ... + p_{\bar{m}_A} = f_i \cdot (z_i{}^{\bar{m}_B} + q_1 z_i{}^{\bar{m}_B - 1} + ... + q_{\bar{m}_B}) \quad (3.8)$$

la cual genera un sistema de $\bar{m}$ ecuaciones, que puede resolverse a través de eliminación Gaussiana. Las soluciones del sistema de ecuaciones corresponden a los coeficientes de los polinomios $P'(Z)$ y $Q'(Z)$ y una vez que se tienen estos polinomios son divididos por su máximo común divisor para eliminar los factores comunes, dejando a los polinomios en una forma con la cual se procede a hallar las raíces de $P'(Z)$ que corresponden a los valores de Δ_A, del mismo modo, las raíces de $Q'(Z)$ indicaran cuáles son los valores de Δ_B.

El orden de complejidad del algoritmo[6] para hallar las raíces de la función racional es de orden $O(\bar{m}^3)$. La cota superior $\bar{m}$ define un máximo número de diferencias entre los conjuntos a reconciliar. Esta consideración es importante para establecer la complejidad de los algoritmos que utilizan esta técnica. Una cota demasiado grande puede hacer que el desempeño del algoritmo no sea el mejor y por otro lado un cota menor al tamaño de la diferencia simétrica no permite establecer la unicidad dada por el teorema (1). Dadas estas consideraciones se espera que la cota $\bar{m}$ sea considerablemente menor que la cardinalidad de los conjuntos a reconciliar pero mayor que la cardinalidad de la diferencia simétrica.

No tener información sobre los conjuntos presenta un obstáculo en el momento de establecer la cota $\bar{m}$. Para resolver este problema se pueden elegir métodos probabilísticos para determinar la cota o incluso la ejecución aleatoria sobre la cota $\bar{m}$ puede servir, aunque el problema de desempeño continua si la diferencia entre los conjuntos es grande. Por este motivo se establecera que en un principio ambos conjuntos calculan sus polinomios siendo iguales y después, en caso de cambiar, mantener sus polinomios utilizando las ideas de actualización de polinomios vistas anteriormente 3.1.1.1.

3.1.3. Ejemplo

A continuación se muestra un ejemplo de reconciliación de conjuntos con la técnica explicada en las secciones anteriores.

Sean dos conjuntos a reconciliar $A = \{1, 2, 3, 4, 5, 6, 7\}$ y $B = \{2, 4, 5, 6, 7, 8\}$, definimos a $\bar{m} = 3$ como la cota de la diferencia simétrica. El campo que se utilizará será $F = 10$, de esta manera se eligen $\bar{m}$ puntos de evaluación, los cuales no pertenecen a los conjuntos A o B, sea el conjunto de puntos de evaluación $Z = \{-1, -2, -3\}$.

Primero se calcula $P(Z)$ el polinomio característico de A:

$$\begin{aligned} P(-1) &= (-1-1)(-1-2)(-1-3)(-1-4)(-1-5)(-1-6)(-1-7) = -40320 \\ P(-2) &= (-2-1)(-2-2)(-2-3)(-2-4)(-2-5)(-2-6)(-2-7) = -181440 \\ P(-3) &= (-3-1)(-3-2)(-3-3)(-3-4)(-3-5)(-3-6)(-3-7) = -604800 \end{aligned}$$

De igual forma se calcula $Q(Z)$ como el polinomio característico de B:

[6] Utilizando eliminación Gaussiana y Factorización de polinomios

$$Q(-1) = (-1-2)(-1-4)(-1-5)(-1-6)(-1-7)(-1-8) = 45360$$
$$Q(-2) = (-2-2)(-2-4)(-2-5)(-2-6)(-2-7)(-2-8) = 120960$$
$$Q(-3) = (-3-2)(-3-4)(-3-5)(-3-6)(-3-7)(-3-8) = 277200$$

Lo cual define la función racional $f(Z) = \frac{P(Z)}{Q(Z)}$, esta función define también los pares (z_i, f_i) del conjunto soporte V, es decir, los pares (z_i, f_i) tales que $f(z_i) = f_i$.

$$f(-1) = \tfrac{-40320}{45360} = -0.\bar{8}$$
$$f(-2) = \tfrac{-181440}{120960} = -1{,}5$$
$$f(-3) = \tfrac{-604800}{277200} = -2.1\bar{8}$$

Es necesario que establecer los grados de los polinomios, para ello se obtiene $d = |A| - |B| = \Delta_A - \Delta_B = m_A - m_B = 7 - 6 = 1$ la diferencia entre los grados del numerador P y el denominador Q. Por otro lado, $\bar{m}$ es una cota superior de la diferencia simétrica, la cual tiene por tamaño a $m_A + m_B$. Debido a esto se pueden acotar los grados del numerador y denominador $\bar{m_A}$ y $\bar{m_B}$ respectivamente:

$$\bar{m_A} = \lfloor (\bar{m} + d)/2 \rfloor = \lfloor (3+1)/2 \rfloor = 2$$
$$\bar{m_B} = \lfloor (\bar{m} - d)/2 \rfloor = \lfloor (3-1)/2 \rfloor = 1$$

Aplicando el teorema (1) se puede asegurar que existe una función racional equivalente, en particular la que se menciona en la ecuación (3.7) que da lugar a $\bar{m} = 3$ restricciones (3.8) como se muestra a continuación:

$$(-1)^2 + p_1(-1)^1 + p_2 = (-0.\bar{8}) \cdot ((-1)^1 + q_1)$$
$$(-2)^2 + p_1(-2)^1 + p_2 = (-1{,}5) \cdot ((-2)^1 + q_1)$$
$$(-3)^2 + p_1(-3)^1 + p_2 = (-2.1\bar{8}) \cdot ((-3)^1 + q_1)$$

De estas restricciones se deriva un sistema de ecuaciones despejando los valores libres del lado derecho:

$$\begin{pmatrix} -1 & 1 & 0.\bar{8} \\ -2 & 1 & 1{,}5 \\ -3 & 1 & 2.1\bar{8} \end{pmatrix} \bullet \begin{pmatrix} p_1 \\ p_2 \\ q_1 \end{pmatrix} = \begin{pmatrix} -0{,}2 \\ -1 \\ -2.4\bar{5} \end{pmatrix}$$

La interpolación se lleva a cabo al resolver el sistema de ecuaciones[7] que da como solución los coeficientes $p_1 = 4$, $p_2 = 3$ y $q_1 = 8$ de los polinomios para finalmente obtener la función racional equivalente $g(Z)$:

$$g(Z) = \tfrac{P'(Z)}{Q'(Z)} = \tfrac{Z^2 - 4Z + 3}{Z - 8}$$

Factorizando los polinomios obtenemos en el numerador $(Z-3)(Z-1)$ y en el denominador $(Z-8)$, lo cual nos indica que las raíces en el numerador son $\{1, 3\}$ correspondientes a los valores en Δ_A y en el denominador $\{8\}$ los valores de Δ_B, los cuales forman la diferencia simétrica. Con estos valores podemos reparar los conjuntos A y B.

[7] Como se menciona utilizando el método de eliminación Gaussiana

3.2. Métrica de inconsistencias mediante reconciliación de conjuntos

La técnica de reconciliación de conjuntos explicada anteriormente tiene como objetivo principal obtener los datos presentes en la diferencia simétrica. Como se menciona en la introducción ésta tesis continua la línea de trabajo en bases de datos distribuidas expuesta en [6, 2]. A continuación se presenta cómo esta técnica ha sido utilizada para obtener una métrica sobre la calidad de los datos de una Base de Datos Distribuida[8].

El trabajo realizado en [6] propone y estudia métricas de inconsistencia de réplicas para Bases de Datos Distribuidas además de cómo calcularlas de manera eficiente y poder comparar su comportamiento. Las métricas propuestas se basan en el cálculo de la diferencia simétrica y para ello se proponen dos técnicas, una de *Traspaso Completo de los datos* [9] y la ya explicada técnica de *Reconciliación de Conjuntos*.

El principal trabajo experimental de evaluación y comparación entre distintas técnicas para el cálculo de la métrica de inconsistencia de réplicas fue realizado en [2], las evaluaciones de diferentes algoritmos para el cálculo de la métrica de inconsistencias de réplicas tomaron en cuenta sólo la llave primaria de cada tabla que contiene los datos. El artículo [6] considera esta cuestión y muestra una versión en la cual la métrica de inconsistencias toma en cuenta todos los datos de una tabla y no sólo las llaves primarias.

En esta seccion se mostrará de manera concisa cómo se utilizó la técnica de reconciliación de conjuntos explicada al inicio del capítulo para calcular la métrica de inconsistencia. A su vez, pretende contextualizar el capítulo siguiente dando un enfoque más orientado al área de bases de datos.

En una base de datos distribuida se tiene un esquema Maestro-Esclavo, sea M el sitio Maestro junto con R_i sitios réplica y T una tabla distribuida bajo este esquema, de esta forma $M.T$ es la copia maestra de T y $R_i.T$ es la i-ésima réplica de T en el sitio R_i.

Una tabla $R_i.T$ cumple la restricción de *consistencia de réplica* [10] si $R_i.T = M.T$, es decir, si es igual a la copia maestra $M.T$.

De igual manera la *consistencia de réplica global* [11] se cumple si todas las réplicas de T son idénticas.

Para saber si una réplica cumple la restricción de *consistencia de réplica* se define:

$$cur(R_i.T) = \frac{|R_i.T \ominus M.T|}{|M.T|} \tag{3.9}$$

Siendo $|R_i.T \ominus M.T|$ la diferencia simétrica, si cur es cero entonces la réplica $R_i.T$ es consistente con la copia maestra.

Por otro lado, para calcular la consistencia global de la tabla T se define otra métrica:

$$gcur(T) = 1 - \frac{|\mathcal{T}_\cap|}{|\mathcal{T}_\cup|} \tag{3.10}$$

[8] Esta sección utiliza definiciones, notación y algoritmos del artículo [6]
[9] Su nombre en inglés: Complete Transfer Aproach
[10] Su nombre en inglés: Replica Consistency
[11] Su nombre en inglés: Global Replica Consistency

Donde $\mathcal{T}_\cap$ es la intersección global de la tabla T y $\mathcal{T}_\cup$ es la unión global. De esta forma se puede medir en qué grado de inconsistencia se encuentra la tabla T globalmente. Si $gcur$ se acerca al número 1 significa que el grado de inconsistencias es muy alto, en caso contrario, cuando $gcur$ es cero no hay inconsistencias, ya que $\mathcal{T}_\cap = \mathcal{T}_\cup$.

Como consecuencia se tiene que si cur es cero para todas las réplicas, entonces $gcur$ también es cero.

En el artículo [6] se proponen formas de cómo llevar a cabo los pasos para obtener el resultado tanto de cur como de $gcur$. Principalmente se basan en el cálculo de la diferencia simétrica, el cuál puede realizarse mediante la técnica de reconciliación de conjuntos.

Para calcular cur se puede proceder como sigue:

Algoritmo 1 Algoritmo para el cálculo de métrica cur

1: Cada sitio calcula su polinomio característico E, esto puede llevarse a cabo al mismo tiempo en cada sitio.
2: El sitio maestro envía su polinomio $M.E$ y la cardinalidad $|M.T|$ a la réplicas R_i.
3: La réplica R_i calcula $cur(R_i) = \frac{|R_i.T \ominus M.T|}{|M.T|}$ con ayuda de la técnica de reconciliación de conjuntos utilizando $M.E$ y $R_i.E$ para obtener la diferencia simétrica.
4: R_i notifica a el resultado de $cur(R_i)$.

De manera similar se calcula $gcur$, sólo que para esta métrica lo que se quiere medir es la cardinalidad de las tablas globales $\mathcal{T}_\cap$ y $\mathcal{T}_\cup$. La manera de hacerlo es la siguiente.

Algoritmo 2 Cálculo de métrica $gcur$

1: Cada sitio calcula su polinomio característico E, esto puede llevarse a cabo al mismo tiempo en cada sitio.
2: El sitio maestro envía su polinomio $M.E$ y la cardinalidad $|M.T|$ a todas las réplicas R_i.
3: Cada réplica R_i calcula $|R_i.T \ominus M.T|$ con ayuda de la técnica de reconciliación de conjuntos utilizando $M.E$ y $R_i.E$ para obtener la diferencia simétrica.
4: R_i envía a M la diferencia simétrica, para este caso se enfatiza que el número de datos en la diferencia simétrica es mucho menor al número de datos de las tablas completas.
5: M recibe todos las diferencias simétricas de cada R_i y crea las tablas globales:

- $\mathcal{T}_\cup = M.T \cup (\bigcup_{i=1}^{i=n} (R_i.T - M.T))$
- $\mathcal{T}_\cap = M.T - (\bigcup_{i=1}^{i=n} (M.T - R_i.T))$

6: M calcula $gcur = 1 - \frac{\mathcal{T}_\cap}{\mathcal{T}_\cup}$

A grandes rasgos éste es el procedimiento para el cálculo de las métricas de inconsistencias, en el capítulo siguiente se estudia con más detalle la aplicación de la técnica de reconciliación de conjuntos para reparar los errores de inconsistencia de réplicas ya que la reparación que se implementará tiene como base los pasos iniciales en estos algoritmos.

Capítulo 4

Implementación del algoritmo de reparación.

Sean dos conjuntos de datos a reconciliar A y B. La técnica de reconciliación para los conjuntos A y B puede encontrar los elementos pertenecientes a la diferencia simétrica $(A \ominus B)$ una vez encontrados pueden manipularse a fin de saber que datos pueden eliminarse o agregarse en los conjuntos para sincronizarlos, desde otro enfoque esto puede verse como una **reparación** de los conjuntos. El proceso de reparación o sincronización de los conjuntos A y B debe operar los conjuntos de manera que A sea igual a B, una manera de hacer esto es hallar el conjunto $A \cup B$. Una vez calculada la diferencia simétrica es posible encontrar esta unión de la manera siguiente:

Considérese U el Universo donde $A \in U$ y $B \in U$ y $A \ominus B = (A - B) \cup (B - A)$
Se calcula el conjunto $A \cup ((A - B) \cup (B - A))$

$$
\begin{array}{lcl}
= & (A - B) \cup (A \cup (B - A)) & (Asoc.\cup) \\
= & (A - B) \cup (A \cup (B \cap A^c)) & (Def.-) \\
= & (A - B) \cup ((A \cup B) \cap (A \cup A^c)) & (DeMorgan.\cap) \\
= & (A - B) \cup (A \cup B) & (Def.Univ.)(\cap U) \\
= & (A \cap B^c) \cup (A \cup B) & (Def.-) \\
= & (A \cup (A \cup B)) \cap (B^c \cup (A \cup B)) & (DeMorgan\cup) \\
= & (A \cup B) \cap U & (Def.Univ.) \\
= & A \cup B & (\cap U)
\end{array}
$$

Por otra parte $B \cup ((A - B) \cup (B - A))$

$$\begin{aligned}
&= && (B - A) \cup (B \cup (A - B)) && (Asoc.\cup) \\
&= && (B - A) \cup (B \cup (A \cap B^c)) && (Def.-) \\
&= && (B - A) \cup ((B \cup A) \cap (B \cup B^c)) && (DeMorgan.\cap) \\
&= && (B - A) \cup (B \cup A) && (Def.Univ.)(\cap U) \\
&= && (B \cap A^c) \cup (B \cup A) && (Def.-) \\
&= && (B \cup (B \cup A)) \cap (A^c \cup (B \cup A)) && (DeMorgan\cup) \\
&= && (B \cup A) \cap U && (Def.Univ.) \\
&= && B \cup A && (\cap U)
\end{aligned}$$

Es decir $A \cup (A \ominus B) = B \cup (A \ominus B)$, siendo ésta una manera en la que podemos establecer una igualdad entre los conjuntos A y B. Lo que se hace es agregar en A los elementos no comunes con B y de igual manera, agregar a B los elementos que no están en A.

$$A = A \cup (B - A) \tag{4.1}$$

$$B = B \cup (A - B) \tag{4.2}$$

Como se explicó en el capítulo anterior, la técnica de reconciliación de conjuntos propuesta para este trabajo nos permite conocer por separado los valores de la diferencia simétrica[1]. Gracias a esta propiedad, se pueden realizar los cálculos de las ecuaciones (4.1) y (4.2) de manera directa pero teniendo en cuenta que los conjuntos A y B pueden estar en diferentes sitios.

Una base de datos distribuida (BDD) con esquema maestro-esclavo y replicación de datos asíncrona, almacena conjuntos de datos en cada sitio. Los datos se almacenan en forma de tablas, donde cada fila representa una tupla y cada columna un atributo de la tabla.

Sea M el sitio maestro y R_i el i-ésimo sitio esclavo. Además sea T una tabla de la BDD, de manera más específica, $M.T$ identificará la copia maestra de la tabla T y $R_i.T$ la i-ésima réplica de T.

Dado que $M.T$ y $R_i.T$ son conjuntos de datos podemos ejecutar ciertas operaciones en cada conjunto para obtener un mismo conjunto de datos ($M.T \cup R_i.T$). La reconciliación de ambos conjuntos es la operación que permite hacer esto como se mostró anteriormente.

Este capítulo presenta cómo se desarrolló la implementación del algoritmo de reparación[2]. Primero se introducen definiciones y observaciones sobre el esquema y modelo de la BDD, se realizan modificaciones a un algoritmo base presentado en el artículo [6] para obtener finalmente una descripción de la implementación y basándose en la descripción se muestra un diseño y se elige un ambiente de desarrollo.

4.1. Reconciliación de conjuntos como reparación de réplicas.

Una BDD que sigue un esquema maestro-esclavo con replicación de datos asíncrona cuenta con una *copia maestra* de los datos almacenada en el sitio maestro. Las réplicas de la copia

[1] Elementos faltantes para cada conjunto

[2] Se utilizará el termino de reparación en lugar de sincronización ya que el enfoque es reparar las inconsistencias en las réplicas de las tablas

maestra se encuentran almacenadas en los sitios esclavos. Este esquema solamente permite actualizaciones por parte del usuario[3] en el sitio maestro el cual propagará las actualizaciones a las réplicas. Después de una reparación de las réplicas respecto a la copia maestra, la inconsistencia en las réplicas debe ser nula.

Como se mostró en la introducción de este capítulo, el resultado de la unión entre la diferencia simétrica con cada uno de los conjuntos que participaron en la operación, nos da como resultado un mismo conjunto de datos. La operación para obtener la diferencia simétrica actúa sobre dos conjuntos solamente, este par de conjuntos pueden ser la copia maestra $M.T$ y una réplica $R_i.T$ de la tabla T.

Se ha mostrado que una forma de sincronizar los conjuntos de datos puede llevarse a cabo como se muestra en las ecuaciones (4.1) y (4.2), sustituyendo los conjuntos por las tablas se tiene lo siguiente:

1. $M.T = M.T \cup (R_i.T - M.T)$
2. $R_i.T = R_i.T \cup (M.T - R_i.T)$

Debido al esquema maestro-esclavo de la BDD el proceso de reparación no debe modificar la copia maestra, por lo cual la operación 1 no puede llevarse a cabo, en su lugar se utiliza una idea similar para lograr tal objetivo. A continuación se propone la siguiente reparación para ser implementada:

Sea $M.T$ la copia maestra de los datos y $R_1.T, R_2.T, ..., R_n.T$ sus réplicas, cada una almacenada en sitios diferentes. Se calculan n diferencias simétricas: $(M.T - R_i.T) \cup (R_i.T - M.T), \forall i = 1, 2, ...n$. Y en cada réplica se realiza la siguiente operación:

$$R_i.T = (R_i.T - (R_i.T - M.T)) \cup (M.T - R_i.T) = M.T \tag{4.3}$$

Cada réplica elimina elementos que no existen en la copia maestra y se agregan los elementos que están en la copia maestra pero que faltan en la réplica. Es decir, se eliminan de la réplica aquellos elementos que sobran y se agregan los elementos que faltan. Ésta forma de reparar las tablas evita modificar la copia maestra.

En el capítulo anterior se habló del algoritmo de reconciliación de conjuntos con una complejidad de comunicación relativamente baja [8]. En el artículo [6] se propone un algoritmo que utiliza la reconciliación de conjuntos para obtener una métrica sobre el grado de inconsistencia de las réplicas. La tesis [2] realiza una instrumentación y comparación del algoritmo presentado en [6] con otros algoritmos para calcular la métrica de inconsistencia de réplicas en una BDD que sigue un esquema maestro-esclavo. La contribución de este trabajo es utilizar la reconciliación de conjuntos en BDD para llevar a cabo la reparación de las réplicas que presenten errores de inconsistencias.

El algoritmo (3.2) de [6] utiliza la técnica de reconciliación de conjuntos propuesta en [8]. Se referirá a este algoritmo como *Algoritmo Métrica*.

En la tesis [2] la instrumentación y comparación de los algoritmos sólo se realiza a través de la llave primaria de cada tabla para determinar cuándo dos tuplas son iguales. El artículo [6]

[3] Usuario se refiere a cualquier ente fuera del SBDD, una persona u algún programa

plantea que el uso de la llave primaria no basta para determinar si dos tuplas son idénticas o no. El problema está ligado a la operación de actualización sobre atributos que no pertenecen a la llave primaria.

Sea $M.T$ la copia maestra de la tabla $T(\underline{pk}, at)$ que contiene $t[pk] = k$, es decir, la tupla t donde pk coincide con el valor de k y además $t[at] =$'a'. Primeramente se supondrá que la i-ésima réplica $R_i.T$ es consistente respecto a la tupla t, es decir, $\pi_{at}(\sigma_{pk=k}(M.T)) = \pi_{at}(\sigma_{pk=k}(R_i.T))$. Si en la copia maestra $M.T$ se lleva a cabo una actualización, digamos $t[at] =$'b', entonces la réplica $R_i.T$ tendría una inconsistencia en la tupla t como se puede ver en la Tabla (4.1).

t	$\underline{pk}$	$a1$
$\sigma_{pk=k}(M.T)$	k	'b'
$\sigma_{pk=k}(R_i.T)$	k	'a'

Tabla 4.1: Inconsistencia generada después de una actualización.

Si se utiliza únicamente la llave primaria para determinar la inconsistencia entonces se omiten todos los errores producidos por actualizaciones a atributos que no son llave. Dado que esto aún representa una inconsistencia entre la copia maestra y la réplica, la reparación de las réplicas debe detectar los errores en cualquier atributo de la tupla. El trabajo realizado en [2] realiza la detección de errores en la clave primaria, por lo que el error de actualización expuesto no se considera en el trabajo mencionado.

Debido a la observación anterior, el artículo [6] plantea que una actualización puede ser tratada como una eliminación de toda la tupla t donde $t[pk] = k$ y $t[at] =$'a' seguida de la inserción de la tupla actualizada con los valores $t[pk] = k$ y $t[at] =$'b'. Cabe señalar que esta forma de interpretar una actualización es utilizada sólo para entender y manejar la consistencia a en toda la tupla y que no se planteó modificar la forma en la que el SMBD realiza las actualizaciones.

Para poder detectar las inconsistencias en cualquier atributo se baso el trabajo en la observación del artículo [6] y se consideró como un único valor a la tupla completa, es decir, la clave primaria concatenada con todos los atributos. De esta manera dos tuplas serán distinguibles dependiendo del valor de todos sus atributos.

En la tabla (4.1), actualizar t en $M.T$ puede interpretarse como la eliminación de toda la tupla $t[pk|at] = k$'a' seguida de la inserción de una *nueva* $t[pk|at] = k$'b'. De esta forma la réplica $R_i.T$ tendrá la inconsistencias $t[pk|at] = k$'a' ya que su valor en la copia maestra $M.T$ es $t[pk|at] = k$'b'. Desde este enfoque se puede observar que los elementos contenidos en la diferencia simétrica entre las tablas y sus réplicas serán **tuplas completas**.

A continuación se presentan modificaciones al *Algoritmo Métrica* (3.2) con la finalidad de obtener un algoritmo de reparación entre una tabla y su réplica en una base de datos distribuida que sigue un esquema maestro-esclavo asíncrono.

Algoritmo 3 Algoritmo de reparación de una réplica para una sola tabla.

Definiciones: -

T: Tabla a sincronizar.
M: Sitio que contiene la copia maestra de T.
R: Sitio esclavo que contiene una réplica de T.
$M.T$: Tabla T en el sitio M.
$R.T$: Tabla T en el sitio R.
Tcpu: Tiempo de cómputo.
Tcom: Tiempo de comunicación.

1: (*Antes de ser necesaria la reparación*)
Ambos sitios acuerdan un máximo número de inconsistencias entre la copia maestra y su réplica, digamos d', también acuerdan los mismos d' puntos de evaluación. El valor de d' representa una cota superior del tamaño de la diferencia simétrica. En cada sitio se calcula el polinomio característico y se almacena en el vector de evaluación E formado por la *representación numérica* de las **tuplas completas** de T en cada uno de los d' puntos evaluación. La cardinalidad de T, $|T|$, también se guarda.

2: (*Tcom*)
El sitio M envía el vector de evaluación del polinomio característico $M.E$ y la cardinalidad de la tabla $|M.T|$ al sitio R.

3: (*Tcpu*)
El sitio R recibe $M.E$ y $|M.T|$ y junto con $M.E$ y $|R.T|$ interpola una función racional reducida y calcula la diferencia simétrica entre las tablas $M.T$ y $R.T$, es decir, $(R.T - M.T) \cup (M.T - R.T)$. Se procede a pasar los elementos de la diferencia simétrica (**tuplas completas**) de su *representación numérica* a su *representación original*.

4: (*Tcpu*)
Una vez calculada la diferencia simétrica en R, se ejecutan las operaciones siguientes en R:

- Eliminar la tupla t en $R.T$ tal que $t \in (R.T - M.T)$
- Insertar la tupla t en $R.T$ tal que $t \in (M.T - R.T)$

En el primer paso del algoritmo 3 es importante recalcar que el proceso para encontrar la diferencia simétrica[4] (paso tres) es un proceso de cálculo numérico, así que es necesaria una representación numérica de los datos con la cual se debe trabajar, tal representación numérica se explicará en las secciones posteriores. Como se ha visto el tiempo del cálculo de los polinomios puede omitirse como tiempo de reparación, ya que se puede realizar con anterioridad y mantenerse actualizado eficientemente, de manera que al requerirse la reparación, los polinomios característicos y las cardinalidades de la tabla, asi como en la réplica, ya hayan sido calculados.

El segundo paso es el único que cuenta con tiempo de comunicación, como se explicó en el capítulo anterior, la complejidad de comunicación depende del tamaño de la diferencia simétrica o en este caso la cota d', que define el número de puntos de evaluación con los cuales se calculan los polinomios. Cuando la reparación es requerida, éste es el primer paso del proceso, por ello se deberá considerar su tiempo de ejecución.

[4] Utilizando la técnica de reconciliación de conjuntos del capítulo anterior

El tercer paso es el de mayor complejidad en el algoritmo ya que de él depende encontrar la diferencia simétrica mediante un proceso de cálculo numérico. La complejidad de este paso depende también de la cota d' sobre el número de elementos de la diferencia simétrica.

El cuarto paso realiza la reparación de la réplica con dos tipos de operaciones, primero con operaciones que eliminan las tuplas que están en la réplica pero no se encuentre en la copia maestra. Como consecuencia de ello en la réplica se borran tuplas que han sido eliminadas en la copia maestra (y cuya réplica aún las contienen). También se eliminan tuplas cuya actualización (en cualquier atributo) en la copia maestra aún no se propaga a la réplica. Después las operaciones de inserción añaden en la réplica las tuplas que están en la copia maestra pero no figuran aún en la réplica, éstas pueden ser nuevas o tuplas que ya existían pero fueron modificadas en la copia maestra.

Basándose en el algoritmo (3) se puede realizar la reparación de todas las réplicas y todas las tablas de la BDD, es por ello que la implementación estará basada en el algoritmo (3).

4.2. Diseño de la implementación

En esta sección se propone un diseño para la implementación del algoritmo (3). Como se puede observar cada paso del algoritmo representa un objetivo concreto. El diseño también toma en cuenta los aspectos relacionados con software, hardware y las cuestiones técnicas de la programación.

4.2.1. Definición de módulos

El diseño está basado en dividir cada uno de los objetivos principales en módulos y de esta manera definir cómo interaccionan los módulos para lograr la reparación deseada. En cada módulo se especificarán las tareas que deberán realizarse para lograr su objetivo, también se mencionaran los datos de entrada necesarios, las funciones que proporcionará y el modo de interacción con los otros módulos. En cada módulo se muestra un diagrama de clases con el que se realizará la implementación. La implementación estará dividida en cuatro módulos principales, cada uno de éstos se encargara de un paso del algoritmo (3). También se contempla un módulo de soporte que permite mantener datos comunes entre todos los módulos de manera que se tenga una mejor interacción. El diagrama de paquetes se muestra en la figura (4.1)

4.2.1.1. Módulo Soporte

De manera que cada módulo pueda tener una interfaz común respecto a los parámetros de replicación así como los relacionados con el esquema a replicar, se define un módulo que permite obtener los parámetros a partir de archivos XML. A su vez este módulo contiene dos paquetes que abstraen, por una parte, el modelo de replicación y los datos para llevar a cabo la reparación y por otro lado los datos sobre el esquema lógico de la base de datos a replicar.

- Replicación: Define clases para almacenar los parámetros de la base de datos replicada, sus sitios y los datos de conexión a la base de datos (4.2).

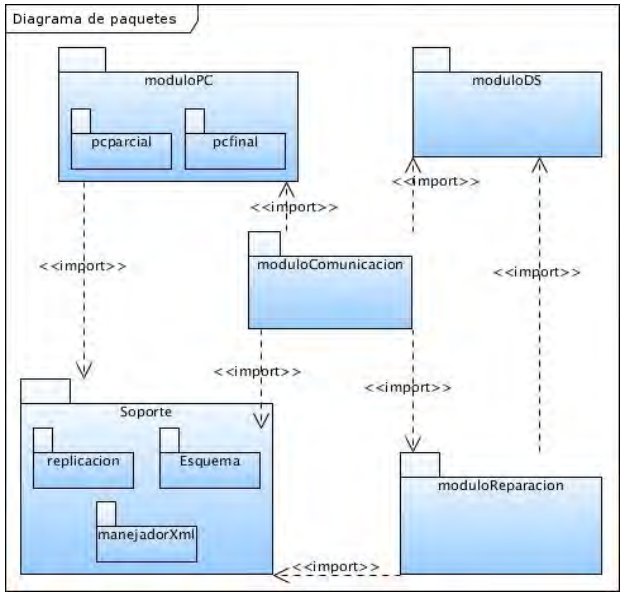

Figura 4.1: Paquetes definidos para la implementación

- Esquema: Define clases para abstraer el esquema lógico de la base de datos, tablas y atributos.(4.3)

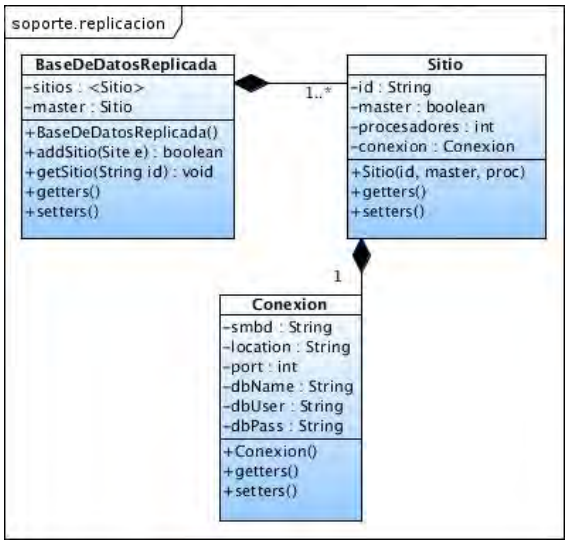

Figura 4.2: Módulo de soporte para la replicación

Como se puede ver en la figura (4.1) este paquete es utilizado por los módulos de cálculo del polinomio, comunicación y de reparación. El esquema definido proporciona información importante para los otros módulos. Uno de los parámetros más importantes se encuentra en la clase **Atributo**(4.3) y es el campo **long** el cual especifica para cada atributo cuál es el tamaño máximo de los valores que puede almacenar, ésto servira para poder reconstruir las tuplas después de obtener su valor original a partir de su representación numérica.

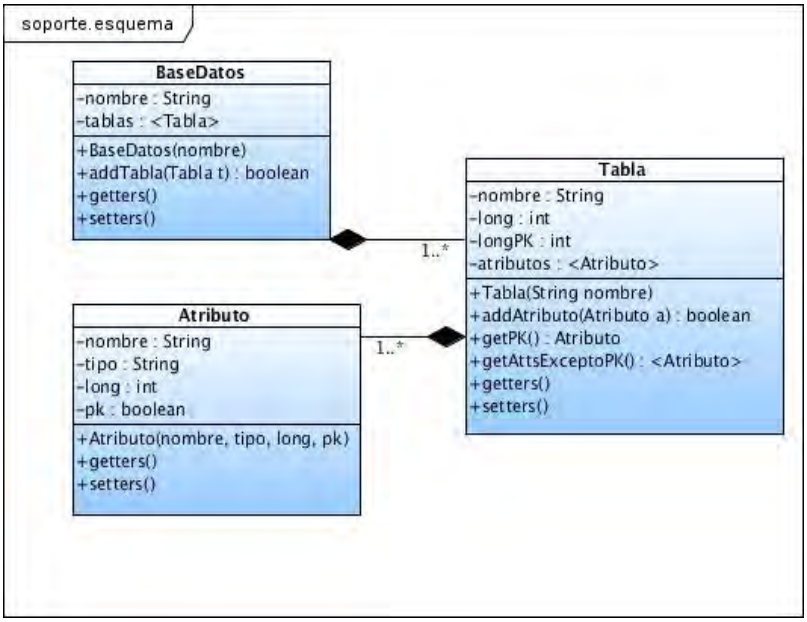

Figura 4.3: Módulo de soporte para el esquema

4.2.1.2. Módulo CalculoPC

Este módulo se encarga de crear los polinomios característicos de las réplicas de las tablas de la base de datos. Los polinomios característicos son la base del funcionamiento del algoritmo. Dado que el cálculo de la diferencia simétrica es un cálculo numérico el módulo tendrá que mapear los datos a un campo numérico y así calcular el polinomio característico. La función o procedimiento de mapeo de los datos deberá ser reversible, es decir, una vez que se tenga el mapeo numérico de un dato, se debe poder obtener el dato original. El mapeo numérico propuesto es un mapeo código de caracteres como pueden ser los códigos ASCII, UNICODE, UTF-8, etc.

Los datos o parámetros de entrada para este módulo son:

- Esquema de la tabla: Permite conocer el tamaño de las tuplas así como los tipos de datos que contiene.
- Sitio donde está la réplica de la tabla: Para conocer en que equipo de la red se encuentra la réplica y como se puede acceder a ella.
- Cota sobre el número máximo de diferencias permitido: En la que se basará la técnica de reconciliación de conjuntos.
- Directorio donde se almacenará el polinomio característico de la tabla.

Dado que es el módulo de preparación se deberán de tener las siguientes funcionalidades:

1. Definir un campo $\mathcal{F}$ con el cual se realizará el cálculo del polinomio.
2. Acceder a los datos de la tabla en la base.

3. Mapear los datos a un campo numérico.
4. Calcular los polinomios característicos.
5. Proporcionar los polinomios característicos y las cardinalidades de la tabla.

Debido a que el mapeo numérico da lugar a números con longitud de dígitos grande se necesita que los cálculos sean eficientes con números grandes. Como se mencionó en la sección anterior la detección de inconsistencias a nivel de tupla completa puede realizarse si se considera una llave compuesta por todos los atributos de la tabla. Por otra parte, los cálculos se realizarán fuera del SMBD por lo que no se modifica el esquema lógico de la base de datos. Para obtener el mapeo numérico de los datos de la tabla se prosigue como sigue.

1. Se obtiene cada tupla de una tabla dada.
2. Se separa[5] la llave primaria y los atributos restantes.
3. Se obtiene la representación numérica de atributos no llave.
4. Se concatenan las representaciones numéricas de los atributos no llave en un orden específico.[6] junto con la llave primaria, obteniendo así una representación única numérica de **toda** la tupla. Es única ya que contiene la llave primaria original.

En este momento se debe considerar cómo es que la representación numérica de la tupla puede ser recuperada. El problema radica en que se debe saber, dado el mapeo de una tupla, qué parte del número corresponde a qué atributo. Para resolver ésto se necesita saber a priori cual es la longitud máxima que puede tener un dato en cada atributo y en base a ello poder completar con algún caracter especial[7] hasta llegar a la longitud del atributo. En caso de tener campos donde no se pueda establecer un limite en la longitud de los datos que almancena, **esta técnica no es aplicable**, ya que los datos pueden ser tan grandes como el manejador lo permita [8]. Una forma de tratar con estos campos es mantenerlos sincronizados de manera separada con otra técnica. Para ello ésta técnica también deberá permitir al usuario especificar cúales atributos deben considerarse en la aplicación del algoritmo y cuales no, esto se puede realizar al recibir el esquema logico por el usuario en algún formato, como puede ser el XML.

Por ejemplo en la figura (4.4) la tabla T tiene tres atributos `pk`, `l_shipmode` y `l_returnflag`, los datos de `l_shipmode` tienen diferentes longitudes, si se establece que la columna `l_shipmode` puede tener cadenas de hasta 10 caracteres y `l_shipmode` sólo almacena 1 entonces el mapeo se muestra en la figura (4.5).

El mapeo toma un caracter y lo transforma en un número de 4 dígitos. Ya que se sabe cuantos dígitos corresponden a cada atributo podemos separarlos y así saber qué datos corresponden a qué atributo. Al conocer de antemano el número de dígitos que puede contener una tupla y el número de dígitos utilizados para su mapeo numérico se puede establecer un campo $\mathcal{F}$ lo suficiente mente grande para contener todos los datos de la tabla.

[5] Se consideran tablas cuya llave primaria es compuesta por un solo atributo entero, OID
[6] Este orden permitirá posteriormente recuperar los datos y su estructura original
[7] Como el espacio en blanco
[8] Tipos de datos BLOB

pk	l_shipmode	l_returnflag
1	TRUCK	F
2	MAIL	R
3	REG AIR	A
4	AIR	D
7	RAIL	B

Figura 4.4: Datos en una tabla T

100840082008500670075003200320032003200320070
200770065007300760032003200320032003200320082
300820069007100320065007300820032003200320065
400650073008200320032003200320032003200320068
700820065007300760032003200320032003200320076

Figura 4.5: Mapeo numérico con código ASCII

El cálculo del polinomio característico debe recorrer toda la tabla, por lo que si se tienen muchos datos entonces éste recorrido puede ser costoso. Una de las ventajas actuales es que los equipos de computo pueden manejar múltiples procesos concurrentemente. El cómputo paralelo de la lectura de la tabla puede ayudar a hacer más eficiente esta tarea. El uso de hilos de ejecución de los lenguajes de programación es lo que nos permitirá llevar esta tarea a cabo de manera eficiente. Para ello se consideró que una tabla puede fragmentarse de manera horizontal como se vio en la sección 2.2.2 y de esta forma se puede leer asignando a cada hilo de ejecución un determinado conjunto de tuplas.

Hasta ahora la lectura de la tabla puede hacerse de forma más eficiente, pero el cálculo del polinomio es un cálculo que involucra todos los datos de la tabla. De acuerdo a la sección 3.1.1.1 y a la ecuación (3.2) se pueden integrar los polinomios parciales de cada partición horizontal de la tabla mediante una multiplicación uno a uno entre los polinomios. Dado que la lectura así como el cálculo del polinomio se puede hacer en forma paralela, se optará por implementarla de esta manera para poder aprovechar el uso de los hilos de ejecución y mejorar el tiempo que tarda en calcularse el polinomio de una tabla.

Por último necesitamos una estructura que guarde y proporcione a los demás módulos el resultado del cálculo del polinomio característico, esta estructura será un simple archivo, al cual los demás módulos podrán acceder y recuperar el polinomio característico. La figura siguiente muestra a grandes rasgos el proceso para crear el polinomio característico de una tabla.

Dadas las observaciones anteriores se especifica un diagrama de clases (4.10) en el que se basará la implementación. Como se mostró en el diagrama de paquetes (4.1) este módulo es utilizado por el módulo de comunicación el cual se encargará de coordinar la ejecución de los cálculos de los polinomios para las distintas tablas y en las distintas réplicas.

Figura 4.6: Cálculo del polinomio característico

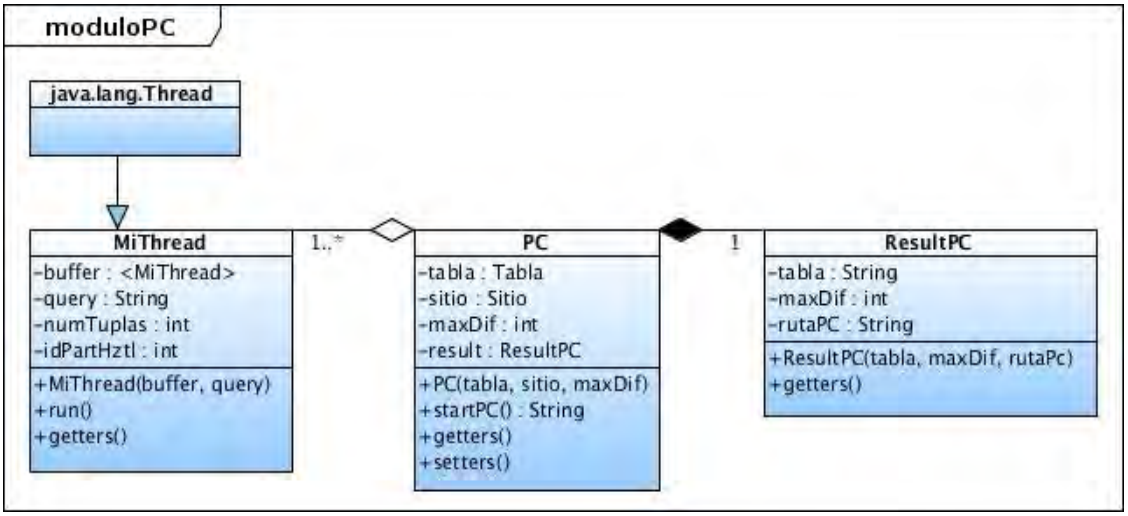

Figura 4.7: Módulo para calcular el polinomio característico

4.2.1.3. Módulo CalculoDS

En el momento en que se requiera reparar las tablas de la base de datos, este módulo será el encargado de realizar los cálculos numéricos que obtendrán la diferencia simétrica con la técnica de reconciliación de conjuntos vista en el capítulo 3. Para ello es necesario que los polinomios característicos de la copia central y de la réplica hayan sido calculados. Este módulo tendrá el proceso de mayor carga computacional de la reparación y se ejecuta en cada sitio esclavo donde se almacene una réplica de la tabla a reparar.

Los parámetros que se deberán proporcionar al módulo son:

1. Polinomio característico y cardinalidad de las tabla de la copia maestra.
2. Polinomio característico y cardinalidad de la réplica de la tabla.
3. Campo u orden utilizado en el cálculo de los polinomios, $\mathcal{F}$.

Las tareas principales que debe realizar este módulo son las siguientes:

1. Realizar la interpolación de una función racional reducida y calcular sus raíces[9]. Las raíces

[9]Factorización del numerador y denominador de la función racional

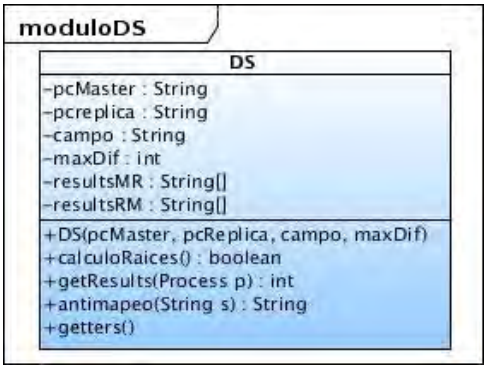

Figura 4.8: Módulo para calcular la diferencia simétrica

precisamente corresponden a los elementos de la diferencia simétrica como se mostró en el capítulo anterior.

2. Obtener los valores originales de los elementos de la diferencia simétrica a través de una función inversa a la función de mapeo numérico.

3. Presentar el resultado, es decir, proporcionar una estructura que almacene la diferencia simétrica.

Este módulo aunque representa un paso importante en la reparación no muestra de manera exhaustiva el proceso de interpolación y obtención de raíces. Ya que este proceso ha sido un problema estudiado e incluso se han hecho implementaciones de estos procedimientos, en particular la implementación hecha en el proyecto CPISync [5] utilizada en este módulo.

El proyecto CPISync [5] fue utilizado también en la parte experimental de [6, 2] donde se utilizaron distintas partes del código para generar un programa que obtiene las raíces de los polinomios. Al utilizar este código, en algunas pruebas generaba resultados erroneos ya que limitaba el uso de números enteros a sólo 32 bits (10 dígitos). Al ser necesario el uso de números extensamente grandes se realizó un análisis del código lo cual llevo a estudiar la biblioteca NTL [1] para resolver el problema, el cual era sólo un problema en la forma de pasar los parámetros al programa, se resolvio este problema y así se logró modificar el código del programa para calcular las raíces con números extensamente grandes[10].

Por otra parte ya que el código utilizado para obtener la diferencia simétrica obtiene resultados aún en su representación numérica, este módulo debe aplicar a los elementos de la diferencia simétrica una función inversa a la del mapeo numérico realizado por el **moduloPC** para obtener los datos antes e haber sido mapeados.

El módulo esta compuesto solamente por una clase que sera la encargada de ejecutar el proceso de obtención de raíces a través del código de CPISync modificado así como guardar el resultado y transformarlo a su representación original.

Este paquete es utilizado tanto por el módulo de reparación como por el módulo de comunicación y no necesita ningún otro paquete para realizar las tareas para las que fue diseñado.

[10] Dado que el mapeo de las tuplas produce números grandes

4.2.1.4. Módulo Reparación

Una vez que el **moduloDS** ha calculado la diferencia simétrica y ha obtenido los elementos en su representación original, según el paso 4 del algoritmo (3) se deben realizar dos tipos de operaciones para completar la reparación: el borrado y la inserción de tuplas de acuerdo al subconjunto de la diferencia simétrica al que pertenecen. Este módulo se ejecutará en cada uno de los sitio esclavos y de esta forma cada réplica estará reparada de acuerdo a los errores individuales que tenia.

Los datos necesarios para llevar a cabo la reparación con este módulo serán:

1. Esquema lógico de la tabla replicada.
2. Datos de conexión a la réplica.
3. Estructura que almacena la diferencia simétrica.

Las funciones de este módulo serán:

1. Obtener las tuplas erróneas y generar las consultas necesarias para la reparación.
2. Eliminar las tuplas que se encuentran en la réplica pero no están en la copia maestra ($R.T - M.T$).
3. Insertar las tuplas que están en la copia maestra y que no están en la réplica ($M.T - R.T$).

La estructura que almacena la diferencia simétrica tiene los datos en su representación original, pero aún se encuentran concatenados por lo que es necesario separarlos basándose en la forma en la que fueron unidos como se explicó en la sección 4.2.1.2 en las figuras (4.4) y (4.5). Esto se lleva a cabo utilizando el esquema lógico de los atributos, en particular la longitud máxima de los datos que puede almacenar cada atributo.

Como ejemplo se tomará un elemento de la figura (4.5) y se expondrá como se genera la consulta para su reparación. Las tuplas se trabajan con su representación numérica, por lo que la tupla t que pertenece a la diferencia simétrica sería:

t	**4**00650073008200320032003200320032003200320068

Al devolver la tupla t a su representación original, en el **moduloDS**, con una función inversa a la del mapeo numérico se obtendría su valor original:

t	4AIR D

Se sabe que el orden de los atributos es `pk`, `l_shipmode` y `l_returnflag`. También se sabe que `l_shipmode` sólo contiene 10 caracteres y que `l_returnflag` sólo 1, por lo cual se puede dividir la tupla t en:

pk	l_shipmode	l_retunflag
4	AIR	D

Si $t \in (R.T - M.T)$ la consulta para la reparación, según el algoritmo (3), sera:

```
DELETE FROM T WHERE pk=4;
```

Pero el caso en que $t \in (M.T - R.T)$ se deberá reparar la réplica insertando los datos de t, para ello debe crearse la consulta:

```
INSERT INTO T(pk,l_shipmode,l_returnflag) VALUES(4,'AIR        ','D');
```

La reparación de la réplica se lleva a cabo al ejecutar las consultas obtenidas como se explicó anteriormente siguiendo los pasos del algoritmo (3). Para este módulo es necesario tener el modelo lógico de la tabla y los resultados del cálculo de la diferencia simétrica por lo que los módulos relacionados son el módulo de soporte y el módulo de cálculo de la diferencia simétrica.

Figura 4.9: Módulo de reparación

4.2.1.5. Módulo Comunicación

Los módulos anteriores están diseñados para ejecutar tareas específicas como parte de un objetivo común, en este caso la reparación. El módulo de comunicación pretende establecer una coordinación entre los módulos para llevar a cabo la reparación. Debido a que las tareas no se realizan en un mismo sitio, ya que los datos están en distintas réplicas, es necesario dividir el módulo en 2 submódulos: un cliente y un servidor. A continuación se exponen las tareas de cada uno de los submódulos y su integración.

Servidor
Este submódulo de comunicación se encargará de atender las peticiones hechas por el submódulo cliente con los recursos del sitio donde se esté ejecutando el servidor (memoria, procesadores o datos). La petición deberá estar especificada correctamente y el servidor deberá devolver sólo una de dos respuestas: verdadero en caso que la petición o tarea se haya realizado correctamente o falso en caso contrario.

El cliente sera diseñado para hacer dos tipos de tareas[11]. Una tarea para el cálculo del polinomio de una tabla y otra tarea encargada de la reparación de una tabla.

[11] En adelante se utilizara el término tarea en lugar de petición

Debido a que el algoritmo (3) requiere que todos los sitios (maestro y esclavos) calculen los polinomios de su copia de datos, el submódulo servidor debe ejecutarse en cada sitio, incluyendo el sitio maestro. Por otra parte el cálculo de la diferencia simétrica así como la reparación sólo se lleva a cabo en los sitios esclavos dado que ellos almacenan las réplicas de la copia maestra. El servidor en el sitio maestro no debe aceptar peticiones de reparación. También se debe asegurar que cada servidor sólo puede acceder a los datos que se encuentran en el mismo sitio donde se esté ejecutando.

Cliente

El submódulo cliente estará encargado de crear y solicitar a un servidor la ejecución de tareas. Como se menciona en la descripción del módulo servidor solamente hay dos tipos de tareas que un cliente debe especificar:

1. Cálculo del polinomio característico de una tabla.
2. Reparación de una tabla.

Dado el esquema maestro-esclavo, el sitio maestro es el que se encarga de propagar las actualizaciones, o desde el punto de vista de este trabajo, reparar los errores. La reparación de los errores será delegada a cada sitio esclavo proporcionando el polinomio característico de la copia maestra almacenada en el sitio maestro. Por esta razón el módulo cliente solamente se ejecutará en el sitio maestro y los esclavos funcionaran a su vez como servidores de tareas.

Como se ha expuesto en los demás módulos, las dos tareas que un cliente puede hacer pueden realizarse mediante la interacción de las tareas ofrecidas por los otros módulos. La tarea de calcular el polinomio característico sera realizada a través del paquete moduloPC. Y la reparación se llevara a cabo mediante los paquetes moduloDS y moduloReparacion. El cliente también debe proporcionar los datos necesarios para cada módulo y debe establecer un medio de comunicación con los distintos Servidores.

El submódulo cliente debe proporcionar los siguientes datos para despues especificar las tareas a realizar para lograr la reparación:

1. Diseño lógico de la base de datos replicada. Proveída por un archivo XML y el paquete **soporte.ManejadorXML**.
2. Datos de localización de los sitios y conexión a la base de datos en cada sitio. Proveída por un archivo XML y el paquete **soporte.ManejadorXML**.

Las funciones que debe realizar:

1. Realizar las distintas peticiones a un servidor para reparar cada una de las réplicas de las tablas de la base de datos distribuida.
2. Obtener el estado de la ejecución de una tarea en el Servidor.

Dado que la reparación debe realizarse en todas las réplicas de los datos, y cada réplica esta situada en un sitio diferente. El cliente debe aprovechar la independencia entre cada réplica

y ejecutar las reparaciones al mismo tiempo. Por esta razón y con tal de hacer más eficiente la reparación, se manejarán hilos de ejecución de tal forma que cada hilo se encargue de la ejecución una tarea en cada sitio. En la figura siguiente se muestra la forma en la que un cliente interactúa con el servidor para realizar una reparación de una tabla.

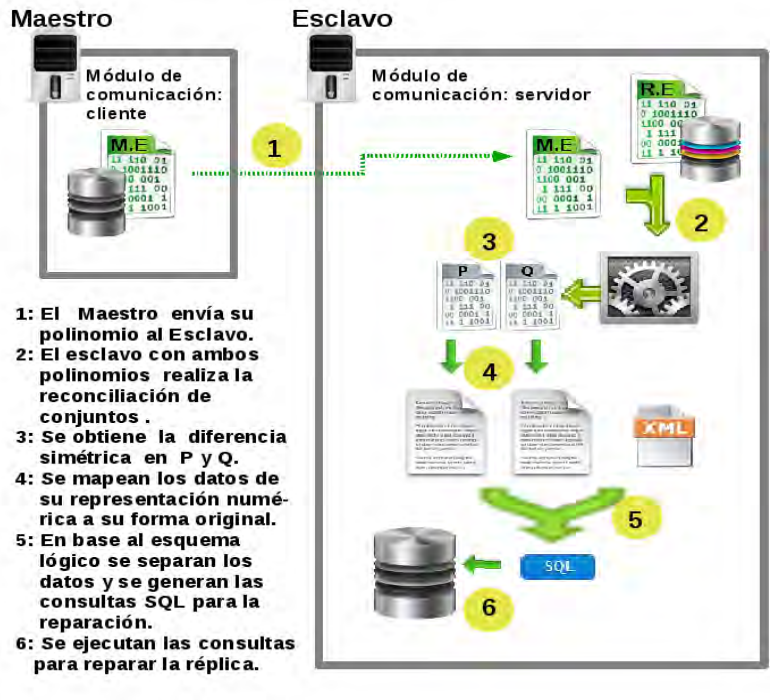

Figura 4.10: Pasos para la reparación de una tabla

La figura (4.11) presenta el diagrama de clases que permite definir la estructura de los submódulos cliente y servidor. El modo de comunicar el cliente con el servidor se diseña como una relación entre las interfaces del servidor y las tareas a realizar. De modo que cada tarea defina la manera de realizar su trabajo, y el servidor realice la ejecución de éstas. El cliente se encargará entonces de obtener una referencia del servidor, se le asignará una tarea y se le pedirá que la ejecute.

4.3. Optimización

Durante el desarrollo de la parte experimental de la sección 5.3.3.2 se observó que la implementación propuesta repara las réplicas pero su rendimiento se ve afectado por el tamaño de datos de las tuplas. Los experimentos proporcionaron información para establecer y considerar una limitante importante de la implementación. Tal limitación está ligada al proceso del cálculo de las raíces para hallar la diferencia simétrica. Este cálculo tiene una complejidad de orden cúbico pero su complejidad también aumenta al utilizar números muy grandes como resultado del mapeo numérico de las tuplas, ésto puede observarse en la figura (5.8). Debido a ello la longitud de las tuplas en la tabla afecta el rendimiento como se muestra en la figura (5.9). A continuación se plantean observaciones sobre los experimentos y se establece un algoritmo que

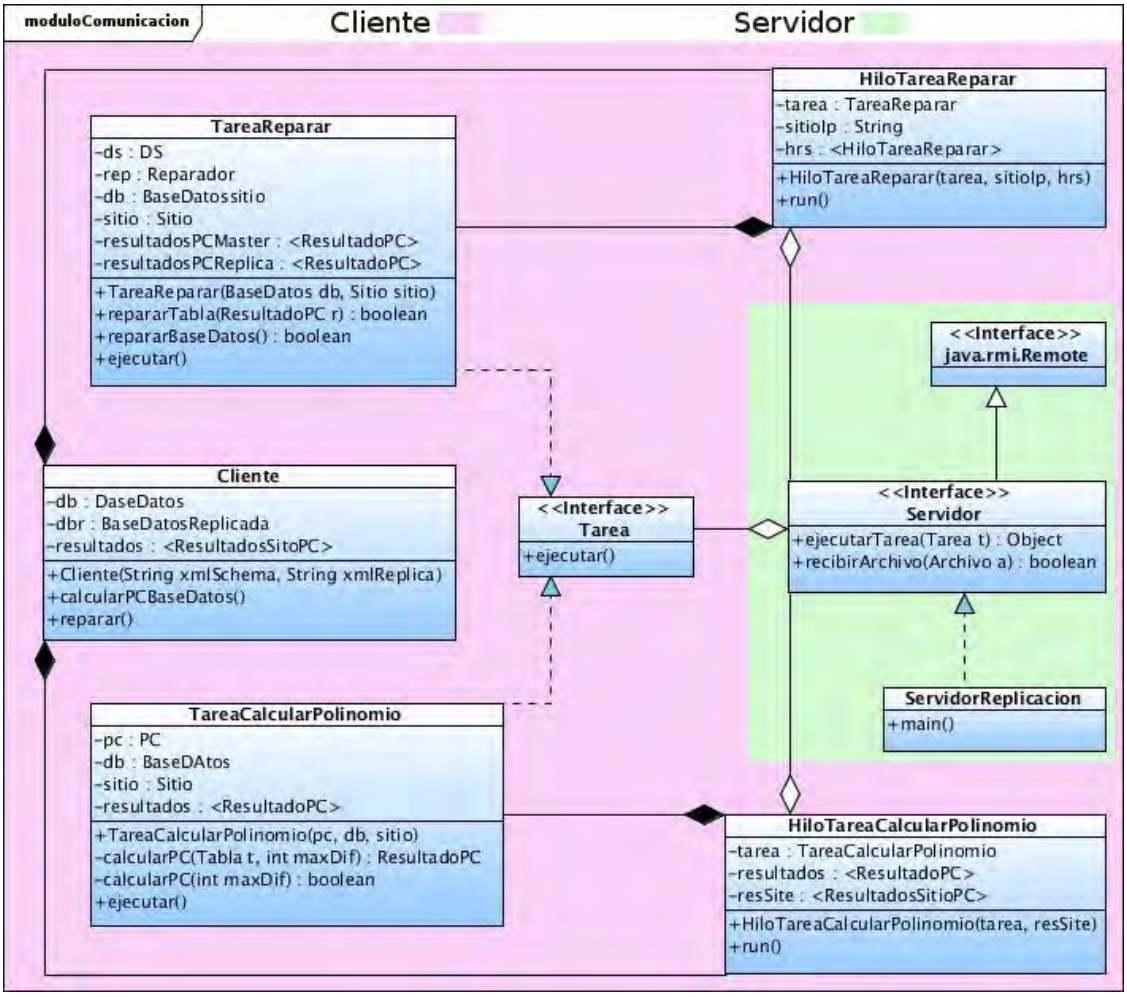

Figura 4.11: Módulo de comunicacion

busca mejorar el tiempo de reparación[12] modificando el algoritmo (3). Por último se muestra el diseño de la implementación del nuevo algoritmo basándose en lo hecho para la implementación descrita anteriormente (4.2).

4.3.1. Limitación en el cálculo de la diferencia simétrica

Como se explicó en el capítulo 3, la técnica de reconciliación de conjuntos utilizada tiene una complejidad cúbica en términos de la cota sobre la cantidad de elementos de la diferencia simétrica. La reparación propuesta consideró manejar un mapeo numérico de las tuplas a fin de poder utilizar esta técnica. Tal mapeo, aunque es efectivo, genera números más grandes en longitud que los datos de la tupla misma. El experimento que compara el comportamiento del algoritmo respecto al tamaño de los números utilizados (5.8) muestra que el aumento de la longitud de los números utilizados influye también en el tiempo del cálculo numérico para la obtención de raíces[13]. Esto a su vez provoca que aunque la cota sobre el número de diferencias

[12] Principalmente el cálculo de raíces
[13] Para obtener la diferencia simétrica, base de la reparación

sea pequeña, si las tuplas en la tabla son muy grandes (el mapeo generará números aún más grandes) entonces el tiempo de cálculo de la diferencia simétrica aumenta considerablemente. De este modo la implementación propuesta hasta ahora tiene 2 limitantes importantes: la cota sobre el máximo número de diferencias y el tamaño de las tuplas en las tablas a reparar.

Dado que la cota sobre el máximo número de diferencias o errores esta ligada fuertemente al problema de reconciliación, se optó por encontrar una solución que permita reparar las tablas aún cuando la longitud de sus tuplas sea grande.

Durante el proceso de experimentación se observó que el cálculo de raíces para obtener la diferencia simétrica hace un uso considerable del procesador, a pesar de esto se notó también que era un solo proceso el que ejecutaba este cálculo, por lo que si la computadora contaba con más de un procesador, el cálculo de las raíces no aprovechaba al máximo el poder de cómputo del equipo, como era el caso.

Basándose en el hecho anterior es como se desarrolló un algoritmo que aprovecha lo más posible los recursos de cómputo para mejorar el cálculo de raíces, hallar la diferencia simétrica y así efectuar la reparación.

4.3.2. Algoritmo de reparación con partición vertical

La forma en la que se pensó aprovechar al máximo el número de procesadores disponibles fue paralelizar el cálculo de raíces. La forma de llevar a cabo la paralelización se basó en otra técnica de fragmentación de una tabla de una base de datos, la fragmentación vertical expuesta en la sección 2.2.1.

Dado que en la fragmentación vertical cada partición tiene un número menor de atributos y por lo tanto de datos, las tuplas de una partición vertical son más pequeñas que la tupla completa. Este hecho generaría que los números a utilizar para el cálculo de la diferencia simétrica también fueran más reducidos.

El algoritmo (3) muestra una reparación entre una tabla y su réplica. Una tabla fragmentada verticalmente puede considerarse como varias tablas replicadas por separado, lo cual si puede resolver la implementación del algoritmo (3).

Tomando en cuenta que cada fragmento vertical de la tabla puede ser considerado como una tabla por separado, los pasos 1,2 y 3 correspondientes al algoritmo (3) no se ven afectados. El problema de la adaptación del algoritmo esta dado por el paso 4 que corresponde precisamente a la reparación.

Para entender mejor el problema que induce el paso 4 del algoritmo (3) considerando una particion vertical de la tabla, se procede a presentar el siguiente ejemplo.

EJEMPLO

Sea la tabla T la que se muestra en la figura (4.12). Como puede observarse esta tabla presenta tres errores de inconsistencia. En la copia maestra las tuplas con llave igual a 1 y 2 no se encuentran en la réplica y de la misma manera, en la copia maestra no se encuentran las tuplas[14] con llave 2 y 4.

[14] Se consideran las tuplas completas

pk	at1	at2	at3
1	a	b	c
2	d	e	f
3	g	h	i

(a): Copia Maestra

pk	at1	at2	at3
2	d	x	f
3	g	h	i
4	j	k	l

(b): Réplica

Figura 4.12: Tabla T antes de ser particionada.

Si se lleva a cabo la siguiente partición de $T(pk, at1, at2, at3)$ en $T_1(pk, at1, at2)$ y $T_2(pk, at3)$ la tabla T queda fragmentada como se muestra en la figura (4.13)

pk	at1	at2
1	a	b
2	d	e
3	g	h

(a): Copia Maestra T_1

pk	at1	at2
2	d	x
3	g	h
4	j	k

(b): Réplica T_1

pk	at3
1	c
2	f
3	i

(c): Copia Maestra T_2

pk	at3
2	f
3	i
4	l

(d): Réplica T_2

Figura 4.13: Tabla T después de fragmentarla verticalmente

Siguiendo los pasos 1,2 y 3 del algoritmo (3), los errores detectados para T_1 son $\{1ab, 2de\} \cup \{2dx, 4jk\}$, mientras que para T_2 se tienen los errores $\{1c\} \cup \{4l\}$. En este punto el paso 4 del algoritmo (3) sugiere hacer las siguientes reparaciones:

1. Eliminar las tuplas 2 y 4 de la réplica de T_1.
2. Insertar las tuplas $1ab$ y $2de$ en la réplica de T_1.
3. Eliminar la tupla 4 de la réplica de T_2.
4. Insertar la tupla $1c$ en la réplica de T_2.

El problema en este paso del algoritmo es que al eliminar las tuplas de una partición se están eliminando las tuplas de las otras particiones ya que se utilizan clausulas DELETE. Este problema se soluciona modificando el modo de reparación de las particiones en las réplicas como se explica a continuación.

Algoritmo 4 Reparación para las particiones verticales de una Tabla.

Definiciones: Cada una de las n particiones T_i de la tabla T ha calculado su diferencia simétrica ejecutando hasta el paso 3 el algoritmo (3).
La diferencia simétrica de la partición T_i será denotada como $(M.T_i - R.T_i) \cup (R.T_i - M.T_i)$.
Sea k el valor de la llave de una tupla t, es decir, $t[pk] = k$. Y sea t_i la i-ésima partición de la tupla $t \in T$.

1: Si $t_i[pk] = k$ y $t_i \in (R.T_i - M.T_i)\ \forall\ i = 1..n$:
La tupla t con llave k en la tabla $R.T$ es eliminada.

2: Si $t_i[pk] = k$ y $t_i \notin (R.T_i - M.T_i)\ p.a.\ i = 1..n$:
No se hace nada.

3: Si $t_i[pk] = k$ y $t_i \in (M.T_i - R.T_i)\ \forall\ i = 1..n$:
Se reúnen todas las tuplas t_i con llave k para formar la tupla completa t, la cual se inserta en $R.T$.

4: Si $t_i[pk] = k$ y $t_i \notin (M.T_i - R.T_i)\ p.a.\ i = 1..n$:
Se reúnen todas las tuplas t_i con llave k y se realiza una **actualización** en la tupla t con llave k de $R.T$.

El paso 1 del algoritmo (4) quiere decir que si en todas las particiones, la tupla t presenta errores, debido a que sus particiones t_i pertenecen a la diferencia $(R.T_i - M.T_i)$, estos errores se deben a uno de dos factores: la tupla esta en la réplica pero no en la copia maestra, o la tupla t en la copia maestra sufrió cambios en todas sus particiones con excepción del atributo que actúa como llave primaria. En cualquier caso la **tupla completa** t no se encuentra en la copia maestra, por lo cual debe ser eliminada.

El paso 2 solamente sugiere que la tupla t presenta errores pero solamente en algunas partes, por lo cual no debe ser eliminada ya que posteriormente sera reparada.

El paso 3 actúa como contraparte del paso 1, en este caso, una tupla t que esta en la copia maestra y que no figura en la réplica provocará que su particion t_i pertenezca a las diferencias $(M.T_i - R.T_i)$ para toda partición de t, por lo cual debe ser insertada completamente. Actúa como contraparte del paso 1 ya que si la tupla t fue eliminada en el paso 1 por tener errores en todas sus particiones entonces es seguro que la inserción de la tupla $M.T.t$ repare esos errores.

El cuarto y último paso también actúa como contraparte del paso 2, si las particiones de t no aparecen en todas las diferencias $(M.T_i - R.T_i)$ es por que sólo algunas de sus particiones presentan error, por lo cual se procede a una **actualización** de las partes inconsistentes.

En este algoritmo se toma como una precondición que el cálculo de las diferencias simétricas para cada partición ya haya sido calculado. El cálculo de estas diferencias simétricas puede hacerse paralelamente ya que como se mencionó al principio de la sección, el proceso de cálculo de raíces[15] solamente se ejecuta en un procesador, al tener un equipo de computo con P procesadores se podrían calcular P diferencias simétricas (cada una correspondiente a una partición vertical de la tabla) paralelamente. Con esta paralelización se busca que el cálculo de las raíces sea más eficiente haciendo que cada proceso se ejecute rápidamente dado que la longitud de los números es menor al tratarse de particiones de la tabla en lugar de la tabla completa.

[15] Y por lo tanto de la diferencia simétrica.

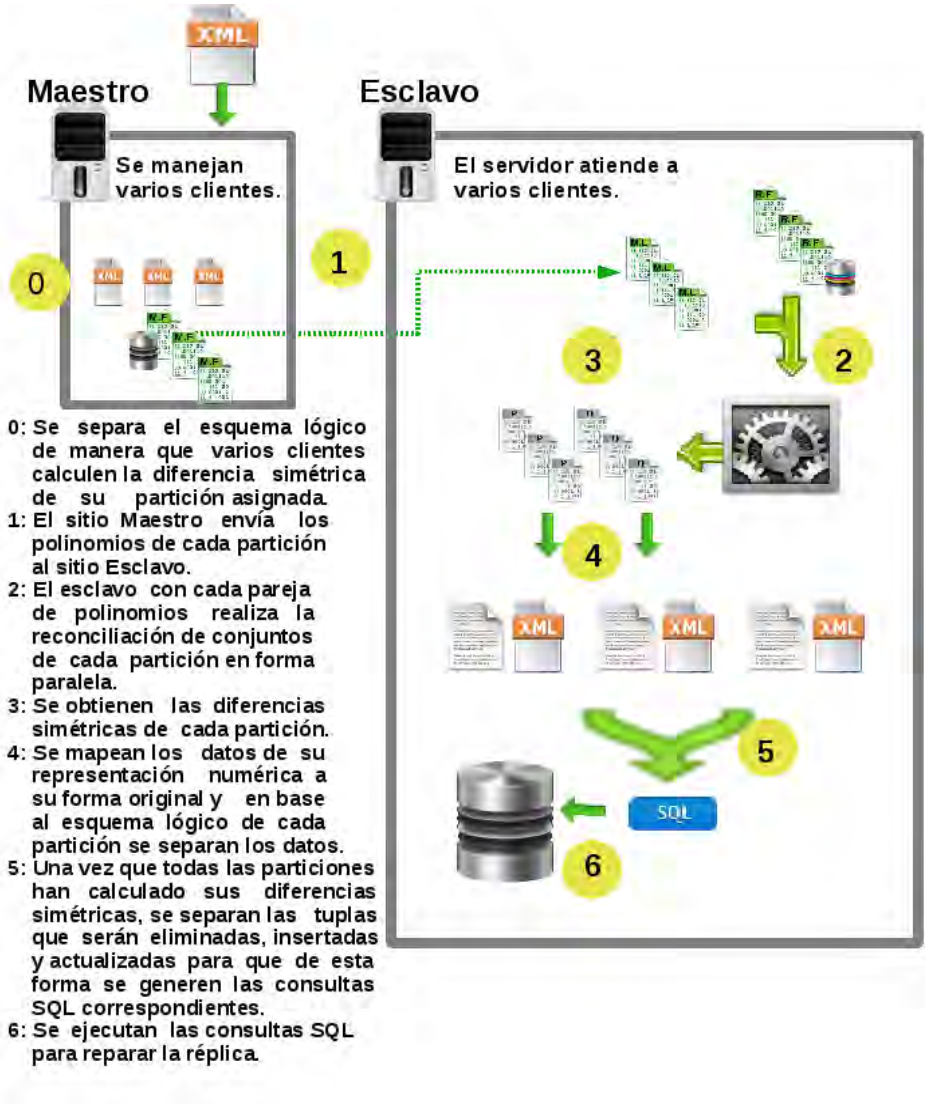

Figura 4.14: Reparación con partición vertical.

4.3.3. Modificación al diseño de la implementación

Como se mostró en el algoritmo (4) se puede utilizar la implementación propuesta en un principio(4.2) para calcular la diferencia simétrica de las distintas particiones generadas por la fragmentación vertical. Para el cálculo paralelo de las diferencias simétricas es necesario desarrollar otro módulo el cual controle esta paralelización. Por otro lado es necesario modificar la reparación ya que además de llevarse acabo de diferente forma se necesita de todos los resultados de las diferencias simétricas. A continuación se presenta un módulo que junto con el módulo de comunicación 4.2.1.5[16] tendrá el objetivo de implementar el algoritmo (4) para su posterior experimentación y comparación con los experimentos anteriores.

[16]El cual ya proporciona el cálculo de la diferencia simétrica

4.3.3.1. Módulo de reparacion vertical

Ya que el cliente tiene que ser ejecutado en el servidor maestro, este módulo también sera ejecutado en el servidor maestro. Y las tareas a realizar en los sitios réplica las realizará a través de la interfaz del cliente.

Este módulo tiene como principal objetivo coordinar los cálculos de las diferencias simétricas en las distintas particiones de la tabla. Cada partición estará asignada a un cliente el cual se encargará de calcular su diferencia simétrica correspondiente.

Para llevar un control de los clientes y obtener el resultado del cálculo de su diferencia simétrica se utilizarán hilos de ejecución. Cada hilo tendrá asignado un cliente al cual se le pedirá que ejecute el cálculo de la diferencia simétrica de su partición asignada. Una vez que el cliente obtenga el resultado, este módulo deberá almacenarlo hasta que el último cliente haya terminado.

Una vez que todos los clientes hayan terminado de calcular las diferencias simétricas se debe realizar el proceso descrito en el algoritmo (4). Es decir, distinguir que tuplas deben ser eliminadas, insertadas o actualizadas en la réplica. Para ello se utilizan tablas hash utilizando como clave el valor de la llave primaria y como valor el resto de la tupla. De esta manera se pueden establecer las condiciones del algoritmo (4). En este punto se aprovecha conocer los valores de las tuplas y la acción a ejecutar para la reparación. Basándose en las funciones del módulo de reparación se generan las consultas DELETE, INSERT y UPDATE necesarias.

Al generar las consultas necesarias para la reparación, estas se le asignan al cliente de manera que él efectúe la reparación en los sitios réplica.

A continuación se presenta el diagrama de clases del paquete descrito:

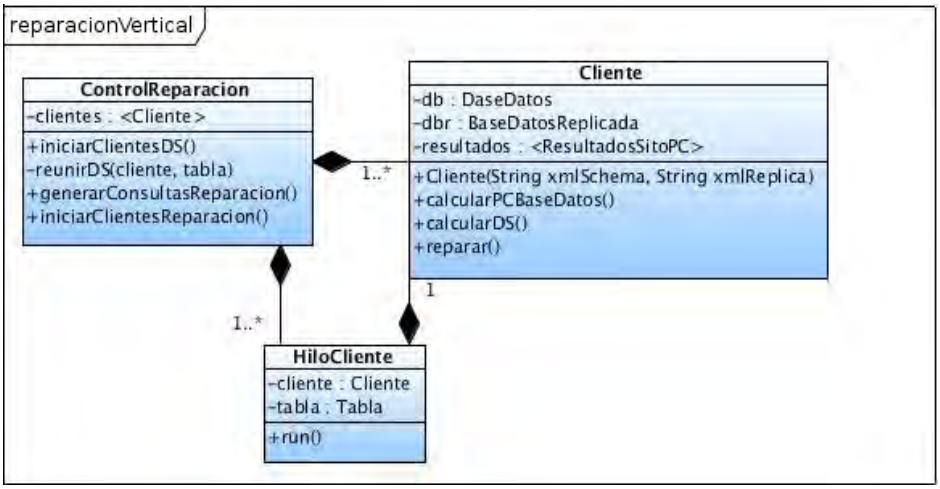

Figura 4.15: Módulo de reparación vertical

4.3.3.2. Manejo de la fragmentación vertical

En la primera implementación se desarrollo un módulo de soporte, en particular el módulo de soporte del esquema 4.2.1.1 que permite obtener el esquema lógico a partir de archivos

XML. Esta característica es una de las más importantes en esta parte ya que su diseño permite fragmentar lógicamente la tabla y de igual manera permite reconstruirla de manera sencilla lo cual ayudo al momento de reunir los resultados de las diferencias simétricas para generar las consultas de reparación.

Dado que el cálculo de la diferencia simétrica es un proceso complejo es necesario que cada fragmento contenga número de datos parecido, es decir, que las tuplas en cada particion tengan una longitud lo más similar posible. De esta manera los cálculos de las diferencias simétricas tendrá un tiempo de ejecución parecido. Esto permitirá aprovechar de mejor manera el tiempo de procesamiento y así disminuir el tiempo del cálculo de la diferencia simétrica total para la reparación.

Una vez que se ha definido una implementación del algoritmo (4) para mejorar el tiempo de reparación se procede a experimentar con ella a fin de comparar los tiempos de reparación contra la implementación anterior. La parte experimental se encuentra en el la última sección del capítulo siguiente.

Capítulo 5

Experimentación

Una vez que se ha llevado a cabo la implementación descrita en el capítulo anterior y tomando en cuenta las cuestiones prácticas del algoritmo (3) para reparar una base de datos se procede a presentar una serie de experimentos, así como sus resultados, en una base de datos de prueba. La base de datos relacional para esta experimentación es una base generada artificialmente, por lo que al ser instalada en los sitios que funcionan como réplicas sera necesario añadir un grado de error controlado de manera que los resultados de los experimentos confirmen y reparen los errores conocidos e inducidos intencionalmente.

Primero se mencionara el ambiente de hardware y software para el desarrollo de la implementación y en el que se realizaron los experimentos. Posteriormente se mostrará la base de datos utilizada y sus características. Los experimentos tendrán el objetivo de comparar el comportamiento del algoritmo de reparación implementado mientras el número de errores crece.

5.1. Ambiente de desarrollo

Una vez que se definieron los módulos es necesario elegir las herramientas con las cuales se llevará a cabo la implementación y el entorno en el que se trabajará. Entre las herramientas principales se encuentra el SMBD, el sistema operativo, bibliotecas y lenguajes de programación. La elección de cada herramienta se escogió de acuerdo a las necesidades de cada módulo, ya sean del problema original o como consecuencia del diseño.

5.1.1. Sistema operativo y SMBD

El sistema operativo en el que se trabajará sera GNU/Linux Debian 5 con el kernel 2.6.26, la elección de este sistema operativo se basa en la gama de herramientas de programación que ofrece como compiladores, bibliotecas, sistemas de archivos, control de procesos y configuración de la red.

Por otra parte el SMBD utilizado es PostgreSQL en su versión 8.4 [10], como se verá más adelante la implementación de la reparación está poco ligada a un SMBD particular y dado que se trabaja sobre el sistema operativo GNU/Linux se optará por utilizar PostgreSQL.

5.1.2. Bibliotecas y lenguajes de programación

En este caso la elección de los lenguajes de programación esta ligada a las necesidades o requerimientos en cada módulo. Principalmente se necesitan hacer cálculos numéricos con una complejidad considerable, también se necesitan interfaces de conexión a la base de datos, control de procesos, hilos de ejecución y manejo del modelo cliente-servidor. Como se menciona en la sección 4.2.1.3 se utilizará código del proyecto CPISync[5], el cual está escrito en el lenguaje C++ y utiliza una biblioteca de cálculo numérico llamada NTL[1] (Number Theory Library) también escrita en C++.

El lenguaje C++ permite hacer cálculos numéricos con números muy grandes eficientemente utilizando la biblioteca NTL. PostgreSQL ofrece una interfaz de programación para la conexión a una base de datos en PostgreSQL en C y C++. C++ soporta el manejo de procesos, hilos de ejecución y el modelo cliente-servidor pero son complicados al momento de implementar.

Por otro lado Java tiene bibliotecas de cálculo numérico pero nos son igual de eficientes, comparadas por ejemplo con NTL. Existen manejadores de conexiones a bases de datos en Java, JDBC. Las clases necesarias para el control de hilos de ejecución y procesos es fácil de programar en Java mediante el uso de monitores. Con lo que respecta al modelo cliente-servidor, Java proporciona la biblioteca RMI[1] para implementar este modelo de manera sencilla.

Dadas las consideraciones anteriores se utilizarán distintas tecnologías y en lo correspondiente a los lenguajes de programación, se utilizarán tanto el lenguaje C++ como Java, ya que ambos proporcionan características que ayudarán a la implementación de la reparación.

5.2. Ambiente de pruebas

El ambiente de pruebas toma en cuenta los aspectos a considerar en cuanto a hardware, software y base de datos de prueba. Las consideraciones sobre el software incluyen el sistema operativo, el sistema manejador de base de datos y los requisitos de la implementación como lo son los lenguajes y bibliotecas de programación.

5.2.1. Hardware

El hardware en el que se realizaron las pruebas consta de dos equipos de iguales características. El hardware en cada uno de ellos es el siguiente:

- 2 Procesadores de doble núcleo cada uno AMD Opteron
- 4 GB en memoria RAM 677 MHz.
- 200 GB en memoria secundaria (Disco duro).

La conexión de los equipos se realizo mediante una conexión LAN Ethernet con 100 Mbits/s como tasa de transmisión. Las pruebas se realizaron en 2 equipos solamente, teniendo en cuenta que al ser un entorno distribuido el tiempo de reparación total puede verse como el

[1] Por su nombre en inglés: Remote Method Invocation

mayor tiempo utilizado para reparar una sola réplica en un sitio esclavo[2]. Las consideraciones de hardware nos permiten afinar la aplicación para obtener un mejor rendimiento, como en el caso del número de procesadores.

5.2.2. Software

Cada equipo funciona con el sistema operativo GNU/Linux Debian 5.0. Dado que el desarrollo de la implementación se realizó sobre una plataforma Linux se eligió mantener este aspecto para las pruebas. Otra razón por la que se utiliza este sistema es que tiene se pueden monitorear los preoceso de manera sencilla y efectiva.

Como SMBD se utilizo PostgreSQL en su versión 8.4 ya que fue la versión utilizada en el ambiente de desarrollo, otra razón fue que tiene bibliotecas de conexion para ambos lenguajes utilizados.

Para construir e instalar la aplicación se necesitan de distintas bibliotecas y compiladores, los cuales se listan a continuación:

- Compilador de C y C++ gcc, versión 4.3.
- Biblioteca de desarrollo NTL (Number Theory Library), versión 5.4.
- Biblioteca cliente de desarrollo para C de PostgreSQL 8.4.
- Java JDK, versión 6.
- Biblioteca JDBC para PostgreSQL, versión 8.3.

Como parte de los requisitos de la aplicación sera necesario el uso de RMI que proporciona Java. Por lo que cada sitio esclavo debe tener el servidor **rmiregistry** en ejecución.

5.2.3. Base de datos de prueba

Como se menciona en la introducción de este capítulo se utilizará una base de datos construida artificialmente. La base de datos utilizada fue creada y es mantenida por el consorcio TPC[12] dedicado a modelar medidas de desempeño para comparar distintos sistemas manejadores de bases de datos. En particular se utilizará una tabla de la base de datos de TPC-H.

Se eligió esta base de datos ya que al ser orientada a OLTP contiene una gran cantidad de datos y de distintos tipos con los que se puede probar la aplicación. El esquema de esta base de datos es un esquema tipo estrella, la cual tiene como tabla de hechos a la tabla *lineitem*, esta tabla es la mas grande dentro de esta base de datos y es la que se utilizará para las pruebas. El esquema lógico de la tabla *lineitem* se muestra en la figura (5.1).

[2] El sitio mas lento

Figura 5.1: Tabla *lineitem* de TPC-H

Por otra parte es importante mostrar los tipos de datos que utiliza esta tabla ya que se debe establecer los atributos y sus tipos para la experimentación:

- Integer, números enteros de 32 bits.
- Numeric(15,2), números reales.
- Char varchar(N), caracteres y cadenas de caracteres acotadas.
- Date, fechas.

Al crear esta base de datos sintética es posible establecer la escala en cuanto a la cantidad de información generada sintéticamente, por defecto la escala es 1 correspondiente a cerca de 1 GB de información, en esta escala la tabla *lineitem* tiene información que ocupa cerca de 700 MB. Ésta será la escala con la que se ejecutarán los experimentos.

La distribución de la tabla *lineitem* se llevará a cabo en dos sitios, cada uno con su propio SMBD. Los errores a generar serán distribuidos en ambos servidores, maestro y réplica.

En cada sitio las inconsistencias correspondientes serán divididas en errores en la llave primaria y errores en atributos no llave para mostrar la reparación de inconsistencias producidas por actualizaciones en atributos no llave.

Basándose en los parametros de experimentación del artículo [6] el número de errores con los que se experimentará serán 100, 200, 300, 500, 1000, 1500 y 2000.

Para generar los errores en cada servidor se ejecutarán sentencias SQL como DELETE y UPDATE en distintos sectores de la tabla para controlar el número de errores generados. Las sentencias DELETE generarán errores en la llave primaria mientras que las sentencias UPDATE generaran errores de actualización. En la Tabla (5.1) se puede observar que en la tabla

Erores en el maestro			Errores en la réplica			
PK	NO-PK	Total	PK	NO-PK	Total	Total de Errores
1-25	-	25	3000001-3000025	4000001-4000025	75	100
1-50	-	50	3000001-3000050	4000001-4000050	150	200
1-75	-	75	3000001-3000075	4000001-4000075	225	300
1-125	-	125	3000001-3000125	4000001-40000125	375	500
1-250	-	250	3000001-3000250	4000001-40000250	750	1000
1-375	-	375	3000001-3000375	4000001-40000375	1125	1500
1-500	-	500	3000001-3000500	4000001-40000500	1500	2000

Tabla 5.1: Tabla de errores generados en la tabla *lineitem*

lineitem del sito réplica se generan más errores. Esto se debe a que al actualizar datos en la réplica se genera el doble de errores, es decir, las tuplas con error en atributos no llave (NO-PK)[3] en el sitio réplica serán inconsistentes con las tuplas en el sitio maestro, de esta manera cada error en atributos no llave generará 2 errores en la diferencia simétrica.

5.3. Experimentos de reparación

A continuación se presenta la ejecución de algunos experimentos de reparación, en ellos se pretende mostrar como se realiza la ejecución de la aplicación y los resultados en la base de datos distribuida.

5.3.1. Ejemplo de parametros de la aplicación

Una vez instalada la base de datos de pruebas y con el porcentaje de errores inducidos, se proceden a establecer los parametros necesarios para que la aplicación pueda llevar a cabo la reparación. Los parametros son establecidos mediante dos archivos XML. EL primero define los parametros de distribución de la base de datos (5.2), y el segundo el esquema lógico de la base de datos (5.3).

Por otra parte hay que definir el esquema de replicación, esta opción hace mas flexible la reparación ya que nos permite definir especificamente qué tablas y atributos se quieren replicar, por ejemplo los atributos de comentarios o anotaciones pueden ser poco útiles y de esta manera se pueden omitir. Por otra parte este requisito requiere tener un poco más de información extra acerca de la base de datos, por ejemplo el tamaño de cada campo y su tipo.

Una vez que se tienen estos archivos XML es necesario proporcionarlos junto con las DTD's a la aplicación y posteriormente invocar a los métodos que se encargan de efectuar la reparación:

[3] Columna en la tabla (5.1)

```
<database>
        <site id="central" master="yes" processors="2">
                <connection     smbd="postgres"         location="localhost"
                                dbport="5432"                dbname="tpch1"
                                dbuser="postgres"         dbpass="postgres" />
        </site >
        <site id="replica" master="no" processors="2">
                <connection     smbd="postgres"         location="localhost"
                                dbport="5432"                dbname="tpch2"
                                dbuser="postgres"         dbpass="postgres" />
        </site >
        <synchronization updates-before-sync="17" />
</database>
```

Figura 5.2: Parametros de replicación: bdd.xml

```
<database name="tpch-nation">
    <table name="nation" >
       <attribute name="n_nationkey" type="n" pk="yes" length="10" />
       <attribute name="n_name" type="s" pk="no" length="25" />
       <attribute name="n_regionkey" type="n" pk="no" length="10" />
    </table>
</database>
```

Figura 5.3: Esquema a replicar: nation_tpch.xml

```
public static void main(String[] args) throws Exception {
        String xmlSchema="nation_tpch.xml";
        String dtdSchema="database-replicated-schema.dtd";
        String xmlSites="bdd.xml";
        String dtdSites="database-replication-configuration.dtd";
        Cliente c;
        c=new Cliente(xmlSchema, dtdSchema, xmlSites, dtdSites);
        //Se calculan los polinomios en todos los sitios
        //Maestro y Replica
        c.computeDatabasePCsAllSites();
        //Una vez calculados los polinomios se procede
        //a reparar la replica
        c.repair();
}
```

Figura 5.4: Utilizando la aplicación.

5.3.2. Ejecución

Dados los parametros especificados en los archivos XML, se presenta un ejemplo para reparar una sola tabla.Este ejemplo se realizó con una tabla más pequeña a fin de poder mostrar los efectos de la ejecución por cuestiones de formato de este documento.

En la figura (5.5) se puede ver que la tabla tiene inconsistencias en su llave primaria así como en atributos no llave, la tabla de la izquierda es la copia maestra y la tabla de la derecha es la réplica. Los elementos en el sitio maestro que no se encuentran en la réplica son $\{2, 18, 28\}$. Mientras que las tuplas que están en la réplica pero no en el sitio maestro son $\{3, 18, 22\}$. La tupla con llave primaria igual a 18 representa un error generado por una actualización, ya que como se puede observar en el sitio maestro el atributo `n_name` tiene el valor `CHINA` mientras que su tupla respectiva en la réplica tiene el valor `JAPON`.

```
tpch1=# SELECT n_nationkey, n_name, n_regionkey from nati
on order by n_nationkey;;
 n_nationkey |          n_name            | n_regionkey
-------------+----------------------------+-------------
           1 | ARGENTINA                  |           1
           2 | CUBA                       |           1
           4 | EGYPT                      |           4
           7 | GERMANY                    |           3
           8 | INDIA                      |           2
           9 | INDONESIA                  |           2
          11 | IRAQ                       |           4
          12 | JAPAN                      |           2
          13 | JORDAN                     |           4
          14 | KENYA                      |           0
          15 | MOROCCO                    |           0
          16 | MOZAMBIQUE                 |           0
          17 | PERU                       |           1
          18 | CHINA                      |           2
          20 | SAUDI ARABIA               |           4
          23 | UNITED KINGDOM             |           3
          24 | UNITED STATES              |           1
          25 | ALGERIA                    |           0
          27 | MEXICO                     |           1
          28 | CROATIA                    |           3
(20 rows)

tpch1=#
```

```
tpch2=# SELECT n_nationkey, n_name, n_regionkey from natio
n order by n_nationkey;
 n_nationkey |          n_name            | n_regionkey
-------------+----------------------------+-------------
           1 | ARGENTINA                  |           1
           3 | CANADA                     |           1
           4 | EGYPT                      |           4
           7 | GERMANY                    |           3
           8 | INDIA                      |           2
           9 | INDONESIA                  |           2
          11 | IRAQ                       |           4
          12 | JAPAN                      |           2
          13 | JORDAN                     |           4
          14 | KENYA                      |           0
          15 | MOROCCO                    |           0
          16 | MOZAMBIQUE                 |           0
          17 | PERU                       |           1
          18 | JAPAN                      |           2
          20 | SAUDI ARABIA               |           4
          22 | RUSSIA                     |           3
          23 | UNITED KINGDOM             |           3
          24 | UNITED STATES              |           1
          25 | ALGERIA                    |           0
          27 | MEXICO                     |           1
(20 rows)

tpch2=#
```

Figura 5.5: Tabla con inconsistencias

Una vez que se ejecuta la reparación como se muestra en la figura (5.4) el sitio réplica identifica los errores y procede a generar y ejecutar las consultas que reparen la réplica como puede observarse en la figura (5.6).

```
/usr/local/bin/roots 53187228905420418418508473437513339940830361398213085664529946493095217860604584887712914782038799
39834139412401975338705794646595487324365194792822189473092273993580587964571659678084484152603881094176995594813302284
1436219 20 20 17 /tmp/pc_central_nation[20] /tmp/pc_replica_nation[20]
[11800670072007300780065003200320032003200320032003200320032003200320032003200320032003200320032003200320032003200320032003
0790065008400730065003200320032003200320032003200320032003200320032003200320032003200320032003200320032003200320032003200320032
32003200320032003200320032003200320032003200320032003200320032003200320032003200320032003200320032003200320032003200320049]
[1030067006500780065006800650032003200320032003200320032003200320032003200320032003200320032003200320032003200320032003
08300830073006500320032003200320032003200320032003200320032003200320032003200320032003200320032003200320032003200320032
32003200320032003200320032003200320032003200320032003200320032003200320032003200320032003200320032003200320032003200320050]
Delete from nation where n_nationkey=3;
Delete from nation where n_nationkey=22;
Delete from nation where n_nationkey=18;
Insert into nation(n_nationkey,n_name,n_regionkey) values(18,'CHINA                    ',2);
Insert into nation(n_nationkey,n_name,n_regionkey) values(28,'CROATIA                  ',3);
Insert into nation(n_nationkey,n_name,n_regionkey) values(2,'CUBA                     ',1);
```

Figura 5.6: Mensajes en el sitio réplica

Las últimas líneas indican las operaciones realizadas en la réplica con el fin de repararla y dejar la tabla igual a la tabla alojada en el sitio maestro, la ejecución de las consultas deja a ambos sitios como se muestra en la figura (5.7) donde la tabla ha sido reparada eliminando los errores de inconsistencias en el sitio réplica.

```
tpch1=# SELECT n_nationkey, n_name, n_regionkey from nati
on order by n_nationkey;;
 n_nationkey |          n_name           | n_regionkey
-------------+---------------------------+-------------
           1 | ARGENTINA                 |           1
           2 | CUBA                      |           1
           4 | EGYPT                     |           4
           7 | GERMANY                   |           3
           8 | INDIA                     |           2
           9 | INDONESIA                 |           2
          11 | IRAQ                      |           4
          12 | JAPAN                     |           2
          13 | JORDAN                    |           4
          14 | KENYA                     |           0
          15 | MOROCCO                   |           0
          16 | MOZAMBIQUE                |           0
          17 | PERU                      |           1
          18 | CHINA                     |           2
          20 | SAUDI ARABIA              |           4
          23 | UNITED KINGDOM            |           3
          24 | UNITED STATES             |           1
          25 | ALGERIA                   |           0
          27 | MEXICO                    |           1
          28 | CROATIA                   |           3
(20 rows)

tpch1=#
```

```
tpch2=# SELECT n_nationkey, n_name, n_regionkey from natio
n order by n_nationkey;
 n_nationkey |          n_name           | n_regionkey
-------------+---------------------------+-------------
           1 | ARGENTINA                 |           1
           2 | CUBA                      |           1
           4 | EGYPT                     |           4
           7 | GERMANY                   |           3
           8 | INDIA                     |           2
           9 | INDONESIA                 |           2
          11 | IRAQ                      |           4
          12 | JAPAN                     |           2
          13 | JORDAN                    |           4
          14 | KENYA                     |           0
          15 | MOROCCO                   |           0
          16 | MOZAMBIQUE                |           0
          17 | PERU                      |           1
          18 | CHINA                     |           2
          20 | SAUDI ARABIA              |           4
          23 | UNITED KINGDOM            |           3
          24 | UNITED STATES             |           1
          25 | ALGERIA                   |           0
          27 | MEXICO                    |           1
          28 | CROATIA                   |           3
(20 rows)

tpch2=#
```

Figura 5.7: Tabla reparada

5.3.3. Experimentos y resultados

A continuación se exponen y explican los experimentos realizados para observar el comportamiento del algoritmo de reparación.

Las preguntas que se pretenden responder con los experimentos son las siguientes:

1. ¿Influye el tamaño de los números en los cálculos numéricos?
2. ¿Cómo se comporta el algoritmo respecto al máximo número de diferencias entre la réplica y la copia maestra?
3. ¿Qué proporción del tiempo de ejecución del algoritmo corresponde a la comunicación, cálculo de raíces y reparación por separado?
4. ¿Como influyen los tipos de datos de la base de datos en la complejidad del algoritmo?
5. ¿La implementación es escalable? ¿Qué elementos de hardware o software son necesarios para mejorar su desempeño?

5.3.3.1. Cálculo de raíces

Éste es el proceso fundamental del algoritmo, pero también es el de mayor complejidad. Como se mostró en la sección 4.2.1.2, el mapeo numérico de los datos genera números muy grandes, en los experimentos con la tabla *lineitem* números con casi 800 dígitos.

El cálculo de raíces involucra dos polinomios característicos, una máxima diferencia o cota y un campo $\mathcal{F}$. Los polinomios característicos para esta prueba son calculados con la tabla *lineitem*. La longitud de los datos de *lineitem* y el campo numérico $\mathcal{F}$ definen la longitud de la representación numérica de los datos.

El primer experimento pretende mostrar como se comporta el cálculo de raíces al manejar números de distintas longitudes. Para ello se fijarán distintas longitudes de números y se calcularán las raíces de acuerdo a una misma cota. Se realiza el mismo experimento pero con una cota mayor.

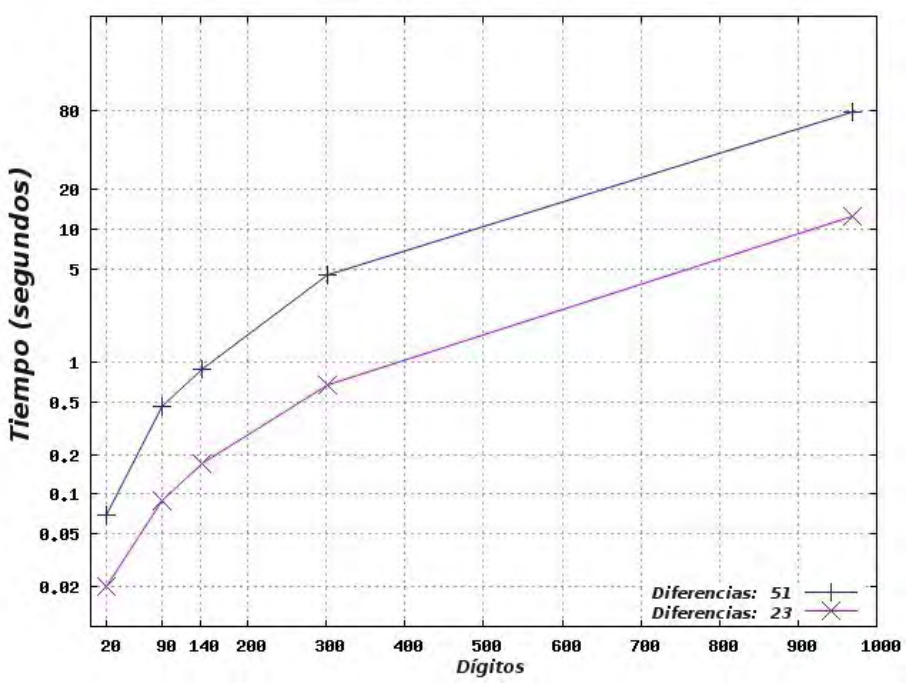

Figura 5.8: Cálculo de raíces variando la longitud de los enteros.

La figura (5.8) muestra los resultados de ambos experimentos, el primero[4] es un experimento en el que se requieren encontrar 20 raíces, en este caso la cota sobre el número de diferencias es 23. El experimento se realizó con números que tenían una longitud de 20, 90, 140, 300 y 970 dígitos. La gráfica tiene una escala logarítmica en el eje Y para mostrar las diferencias entre los valores más pequeños del eje X.

En este experimento puede observarse que conforme se utilizan números de longitud mayor. el tiempo de cálculo de las raíces crece. En el experimento donde se utilizan los números más grandes[5] el tiempo en realizar el cálculo esta alrededor de los 10 segundos, pero debe tenerse en cuenta que el número de diferencias es bajo (23 diferencias).

Para comparar el comportamiento de los números de longitudes variantes respecto al cálculo de raíces se realizó el segundo experimento[6]. De igual manera se utilizaron las longitudes del experimento anterior, pero en este caso se elevo la cota sobre número de diferencias. En la

[4] Línea con taches de la figura (5.8)
[5] Números de 970 dígitos
[6] Línea con cruces de la figura (5.8)

gráfica se puede ver que el comportamiento del algoritmo respecto a la longitud de los números es parecido. La diferencia entre las gráficas representa el tiempo que tardo en calcular mas raíces, 50 en este caso.

De acuerdo a estos dos experimentos se puede establecer que el cálculo de las raíces se ve favorecido por el uso de números de longitud reducida. Contestando la pregunta 1(5.3.3):

Efectivamente, el uso de números de longitud grande afecta de manera considerable el desempeño del cálculo de raíces.

5.3.3.2. Reparación

La sección anterior sólo contemplo el cálculo de las raíces, por lo que es necesario establecer cuál es el comportamiento del algoritmo desde la transmisión del polinomio de la copia maestra hasta la reparación de la réplica.

Para este experimento hay dos variables a considerar, primeramente está la cota sobre el número de elementos en la diferencia simétrica $\bar{d}$ entre la réplica y la copia maestra, la cota mencionada afecta directamente en:

1. El tamaño de los polinomios característicos: Debido a que la cota define el número de puntos de evaluación, entre más grande sea la cota habrá mas puntos de evaluación y el archivo que almacena el polinomio característico crecerá de igual forma.
2. La complejidad del cálculo de las raíces: Como se explicó en el Capítulo 3, el orden de complejidad es $O(\bar{d}^3)$.
3. La cantidad de reparaciones a realizar: Esto es, el total de consultas hechas a la réplica para repararla.

Por otro lado se debe considerar el tamaño de las tuplas en las tablas a replicar, ya que si la tupla tiene muchos atributos y/o cada uno de ellos tienen una longitud grande, el mapeo generará números muy grandes afectando el desempeño en la aplicación como se observó en el experimento anterior (5.8). Para mostrar cómo afecta este fenómeno a la aplicación se realizaron dos experimentos.

El primero[7] establece un conjunto de datos de la tabla *lineitem*, el cual será representado por números de hasta 300 dígitos[8]. Se inducirán los errores controlados y se procederá a repararlos. Posteriormente se realiza el mismo experimento con todos los atributos[9] de *lineitem*, en este caso la longitud de los números alcanza los 970 dígitos. En esta gráfica se puede apreciar de mejor manera el impacto que tiene el uso de números de longitud grande (cercana a los 900 dígitos). La desventaja de no usar números de longitud grande es que se tiene que restringir la cantidad de datos a replicar. En la gráfica de reparación anterior se contempla sólo el tiempo de computo, no así el tiempo de comunicación.

El tiempo de reparación corresponde únicamente a la ejecución de las cláusulas SQL generadas por la aplicación como se muestra en la figura (5.10).

[7] Línea con taches en la figura (5.9)
[8] Nótese que este experimento no contempla toda la tabla *lineitem*
[9] Línea con cruces de la figura (5.9)

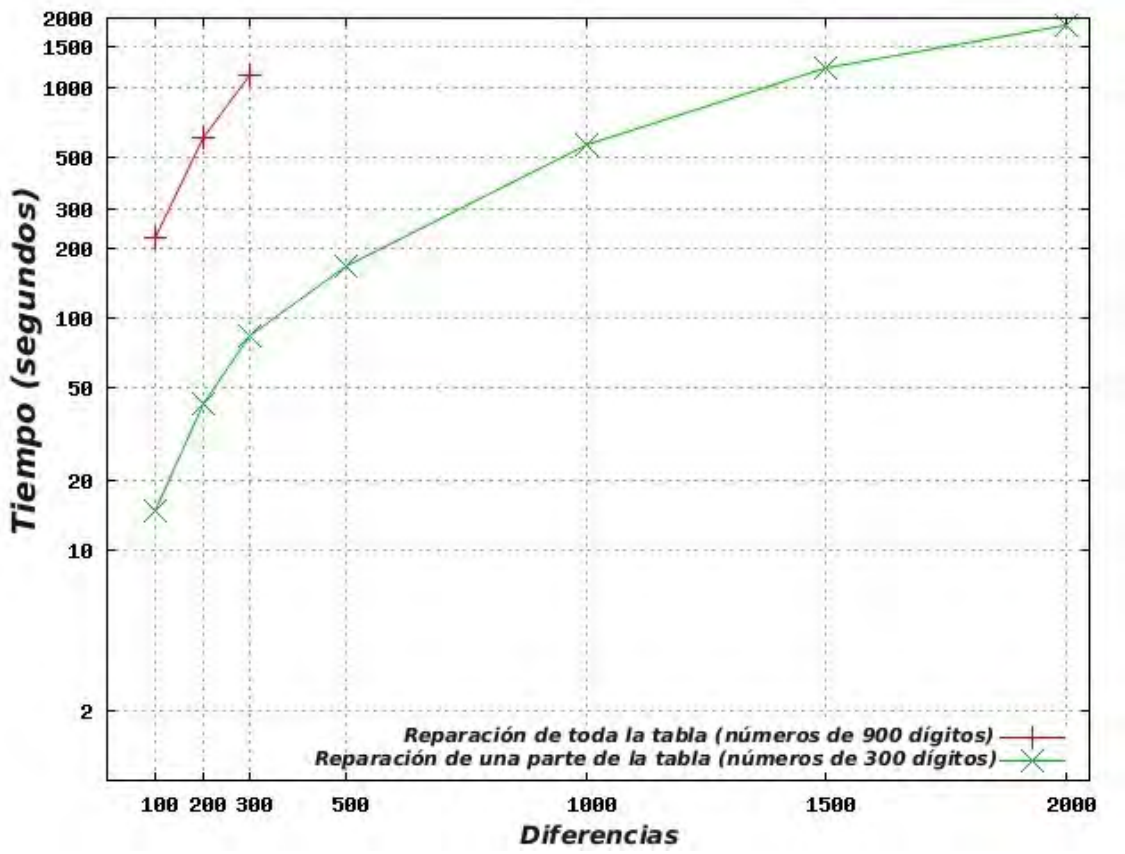

Figura 5.9: Cálculo de diferencia simétrica y reparación en *lineitem*

Debido a que esta reparación se hace mediante la ejecución de consultas SQL su tiempo es muy bajo, menos de 1 segundo en todos los experimentos. Lo único que puede observarse es que el experimento dos[10] ocupó mas tiempo en ejecutar las consultas que el primero[11] de éstos experimentos. Esto se debe a que el experimento 2 se realizo con tuplas más grandes, lo cual provocó que las consultas tardaran un poco más de tiempo en ejecutarse.

El tiempo de comunicación correspondiente al paso de los archivos contenedores del polinomio característico fue muy bajo, el archivo de mayor tamaño que se manejo durante estos experimentos (5.10) no rebaso los 2MB. El tiempo de transmisión de un archivo de 2MB fue medido y este estaba alrededor de 1 segundo. Es de consideración ver que la cantidad de datos transmitido fue reducida drásticamente. Esta es la ventaja significativa del algoritmo de reparación implementado.

Atendiendo a las preguntas(5.3.3) 2, 3 y 4 podemos responder lo siguiente:

- El algoritmo tiene un comportamiento polinomial ya que el tiempo de cálculo de raíces es mucho mayor que los tiempos de reparación y comunicación. Por ende el algoritmo de reparación se comporta de la misma forma que el cálculo de raíces, como se describe en [8], en forma polinomial. Con un orden de $O(\bar{d}^3)$.

[10] Línea con cruces en la figura (5.10)
[11] Línea con taches en la figura (5.10)

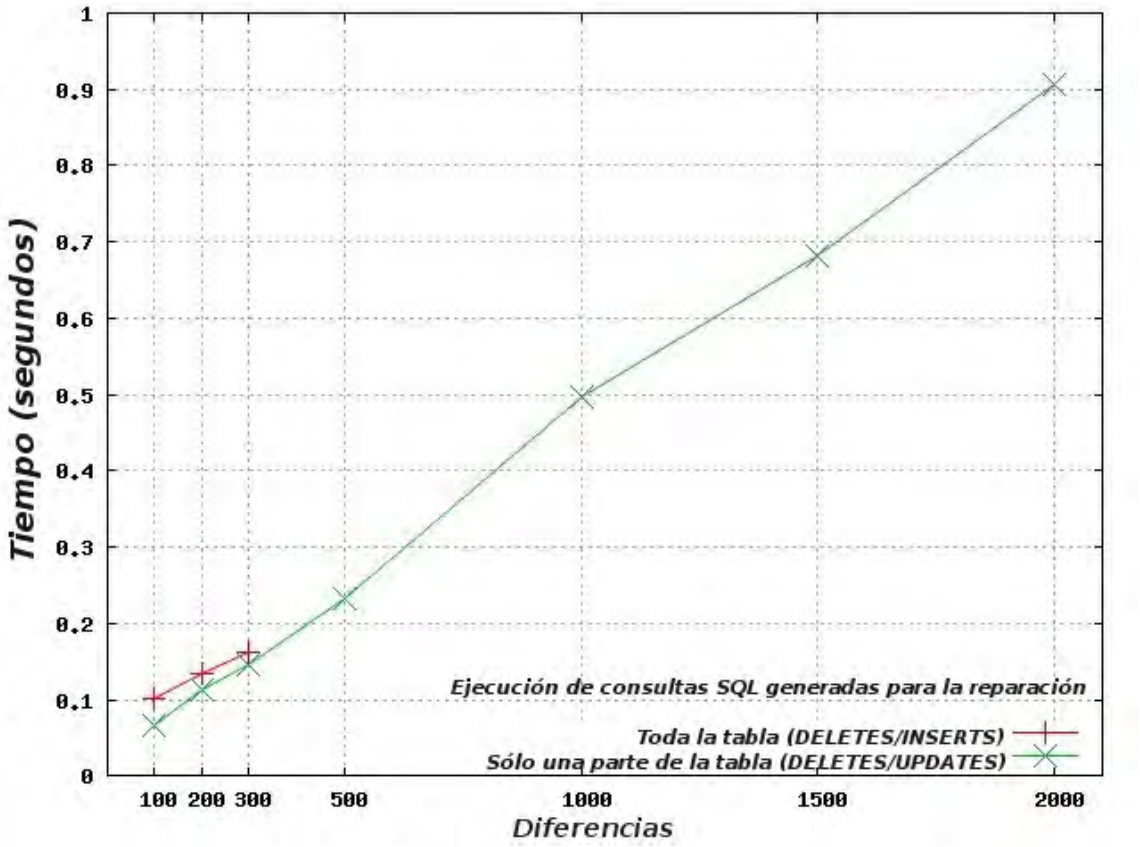

Figura 5.10: Reparación SQL

- Los tiempos de comunicación y reparación son muy bajos, cerca de un segundo cada uno, respecto al tiempo del cálculo de raíces, el cual puede tardar varios minutos. Todo esto dependiendo de la cota $\bar{d}$.

- Los tipos de datos que almacena una tabla son la parte mas importante a considerar para utilizar este algoritmo, ya que se necesita tener información extra, como la longitud máxima que pueden almacenar cada atributo. Aun cuando los tipos de datos tengan una representación reducida[12], la cantidad de atributos de la tabla puede ser grande haciendo que la tupla completa también lo sea. De esta forma la longitud de las tuplas significa un problema importante para el desempeño de la reparación.

5.3.3.3. Reparación vertical

En el Capítulo 4 la última sección plantea una forma de mejorar la reparación basándose en el aprovechamiento de los recursos de computo de los equipos. Una vez que se codificó el algoritmo (4) se procede a experimentar con esta nueva implementación para poder compararla y observar si efectivamente el tiempo de reparación disminuye.

[12] Que no tengan una longitud grande

La pregunta planteada para este experimentos es ¿cómo se comporta la idea de reparación mediante partición vertical frente a la reparación de toda la tabla sin partición[13]?

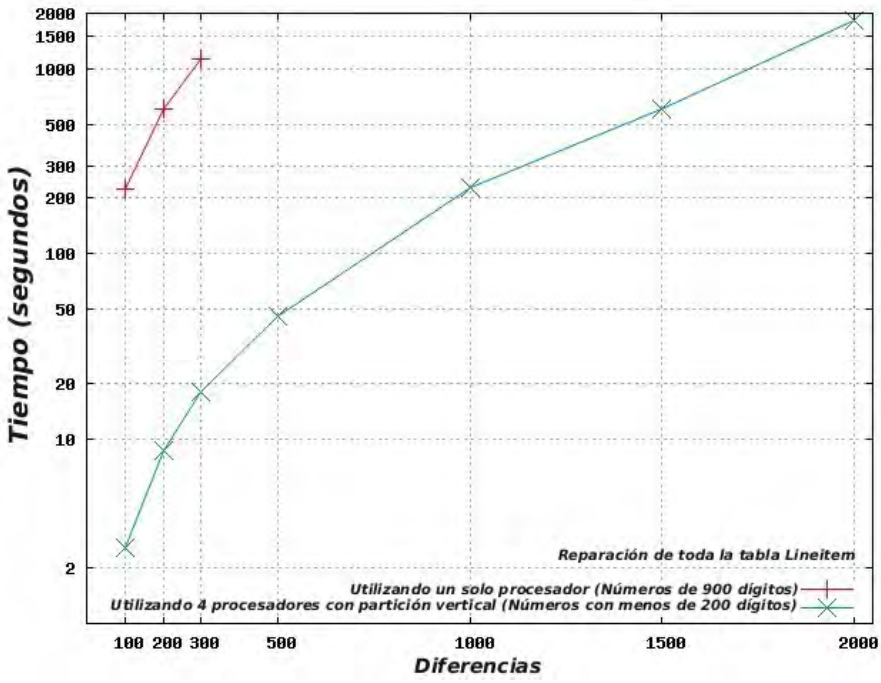

Figura 5.11: Reparación con partición vertical

El siguiente experimento se llevo a cabo con toda la tabla de *lineitem* pero se fragmentó en 8 partes, tratando de que cada particion utilizára números de longitudes pequeñas, se compara con el experimento anterior donde se repara la tabla *lineitem* completa utilizando números de 970 dígitos.

La diferencia es considerable, en este experimento [14] la reparación vertical permitió reparar la tabla completa de *lineitem* en la cual se tenían hasta 2000 errores, mientras que la implementación anterior [15] sólo permitía reparar la tabla completa de *lineitem* hasta con 300 errores. Teniendo en cuenta esta característica se puede establecer que la reparación vertical escala de buena manera en cuanto al número de procesadores. La aceleración[16] que se pudo obtener se presenta en la tabla siguiente. La aceleración no se mantiene, disminuye conforme aumenta la cota sobre el máximo número de diferencias.

Diferencias	Exp_1 (4 procs)	Exp_2 (1 proc)	$Acc_2 = \frac{Exp_2}{Exp_1}$
100	17	223	13.11
200	60	605	10.08
300	130	1135	8.79

Figura 5.12: Aceleración sobre el cálculo de la diferencia simétrica

[13] Como se realizó en los experimentos anteriores
[14] Línea con taches en la figura (5.11)
[15] Línea con cruces en la figura (5.11)
[16] Conocida también como *Speed-up*

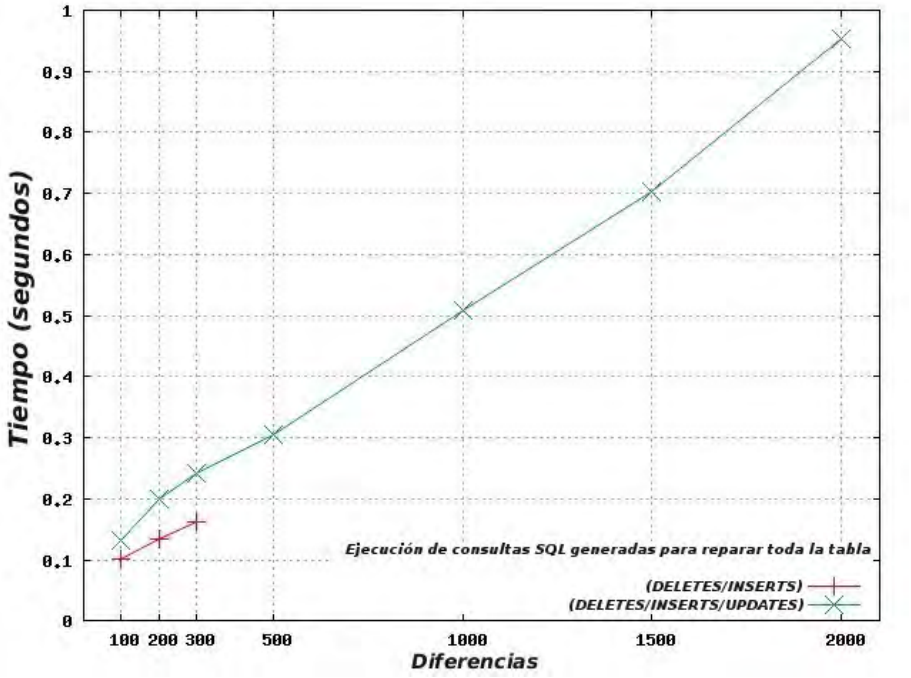

Figura 5.13: Reparación vertical SQL

Con respecto a la reparación a través de las clausulas SQL, la reparación sufre cambios significativos pero se mantiene con un tiempo bajo para su ejecución. La figura (5.13) muestra una comparativa entre los experimentos anteriores y los experimentos de reparación con partición vertical donde la ejecución de las sentencias SQL aumenta debido a que aparte de ejecutar sentencias DELETE e INSERTE se ejecutan también sentencias UPDATE.

De esta forma se da por concluida la parte experimental, el último capítulo expone de manera general una revisión acerca de los objetivos planteados y los objetivos alcanzados a lo largo de este trabajo.

Capítulo 6

Conclusiones

A lo largo de los capítulos anteriores se ha expuesto un algoritmo de reparación de réplicas de una base de datos distribuida utilizando la técnica de reconciliación de conjuntos expuesta también en este trabajo. La característica mas importante de esta técnica es que la complejidad de comunicación no depende del tamaño de los conjuntos, por ello su uso en bases de datos distribuidas puede ser de importancia ya que los sistemas actuales almacenan grandes cantidades de información.

Sobre la técnica de reconciliación de conjuntos es importante mencionar que si bien la complejidad de comunicación no depende del tamaño de los conjuntos de datos, existen otros factores como la cota sobre el número de diferencias o el tamaño de los datos que influyen en el desempeño del cálculo de la diferencia simétrica. De esta manera se necesitó experimentar con esta técnica para poder establecer las condiciones necesarias para su funcionamiento correcto y eficiente.

La adaptación de esta técnica a los sistemas de bases de datos distribuidos implicó un esfuerzo considerable ya que las cuestiones de comunicación y coordinación en la práctica son complicadas. A pesar de ello se logró adaptar esta técnica basándose en los trabajos realizados en las investigaciones relacionadas. Dichas investigaciones ayudaron a obtener una idea más concreta sobre el funcionamiento de la técnica y de las bases de datos distribuidas. Aprovechando este conocimiento se preparó un caso de estudio en el cuál las condiciones necesarias para utilizar la técnica de reconciliación estuvieran presentes y así resolver un problema dentro del caso de estudio como lo es la reparación de errores de inconsistencias de réplicas en una base de datos replicada.

Una vez que se entendieron las principales cuestiones teóricas en ambos problemas[1] se hicieron pruebas y programas que permitieran establecer una interacción entre el software existente[2]. Estos primeros experimentos permitieron establecer los alcances y limitaciones que pudiera tener la aplicación para el caso de estudio. Al estudiar y resolver algunos los problemas de adaptación se pudo generar un diseño de los programas necesarios para llevar a cabo la reparación. La prueba de estos programas permitió observar el comportamiento de la reconciliación de conjuntos reflejado en una base de datos distribuida y con ello también se estableció un plan de pruebas para la experimentación.

[1] La reconciliación de conjuntos y la reparación de réplicas.
[2] Proyecto CPISync y un SMBD como PostgreSQL.

La parte experimental permitió comprobar que la adaptación realizada mantenía el comportamiento de la técnica de reconciliación. Más aun, permitió identificar limitaciones que no se contemplaban en los trabajos relacionados como lo es el aprovechamiento del cómputo paralelo para mejorar el tiempo de cálculo de la diferencia simétrica. Debido a estos nuevos descubrimientos se pudo reutilizar la implementación hasta entonces realizada para construir una versión que permitiera aprovechar de mejor manera los recursos de cómputo. Esta segunda fase de diseño y programación resultó menos laboriosa ya que se tenía una base funcional y que podía ser adaptada de manera sencilla. Posteriormente se experimento la nueva versión para comparar y saber en qué proporción mejoraba la reparación al utilizar más procesadores para el cálculo de la diferencia simétrica. Como se vio en el capítulo de experimentación, la reparación con particion vertical mejora considerablemente por lo que este descubrimiento es una de las partes mas importantes de este trabajo.

La experimentación y observación del comportamiento del algoritmo de reparación que se desarrollo proporciona información que se resume a continuación:

- La complejidad de comunicación del algoritmo de reparación se encuentra determinada por una cota sobre el número de diferencias que hay entre las réplicas y el tamaño de los datos sin importar la cantidad de datos almacenada en las tablas de la base.

- La complejidad de cómputo de la diferencia simétrica y obtención de raíces tiene dos factores que influyen en su desempeño. Uno de ellos es la cota sobre la diferencia simétrica, la cual afecta también la creación de los polinomios. En cuanto al cálculo de la diferencia simétrica y obtención de raíces la cota afecta el desempeño ya que la complejidad de este cálculo es de orden cúbico. El segundo factor es el tamaño de los datos a reparar ya que al manejar datos muy grandes la complejidad de los cálculos numéricos también se ve afectada.

- Aun cuando la complejidad cúbica de la técnica de reconciliación no pudo mejorarse, se pudo mejorar el desempeño al partir los datos y llevar el cálculo de forma paralela, tratando así la limitante sobre el tamaño de los datos mediante paralelismo.

- Para que el uso del algoritmo de reparación sea viable se pueden requerir metadatos acerca de la base de datos replicada como la información de conexión y principalmente información acerca de la estructura de las tablas. Esta información permite recuperar la estructura de los datos para poder generar las cláusulas SQL que realizan las reparaciones.

- La aplicación de esta forma de reparación no es viable si se quieren reparar datos que presentan un tamaño demasiado grande[3] o que no se puedan acotar. Para tratar este tipo de datos se pueden utilizar otras técnicas de reparación o sincronización. Pensando en esta cuestion, la aplicación es flexible al permitir elegir las columnas de las tablas sobre las que se ejecutara la reparación.

- Se observó que en todo el proceso gran parte del tiempo de reparación recae en el cálculo de la diferencia simétrica por lo que al considerar esta forma de reparación frente a otras se debe establecer una condición de comparación entre este cálculo y el tiempo de reparación con otra técnica que se pudiera utilizar.

[3] Grande en cuanto a datos almacenados en un atributo

- La comunicación en este algoritmo también involucra una forma de mantener la seguridad de los datos transmitidos por la red de comunicación ya que los datos son mapeados a un campo numérico $\mathcal{F}$ y además son necesarios ambos polinomios para obtener los datos reales. De esta forma el mapeo de los datos y la interpolación de los polinomios puede funcionar como una encriptación de llave asimétrica, donde las llaves son precisamente los polinomios característicos, el campo numérico y la cota sobre el máximo número de diferencias.

Se puede concluir que la reparación que se ha expuesto y probado a lo largo del trabajo es funcional y tiene características que pueden ser requeridas por los sistemas de bases de datos distribuidas, como la baja comunicación para la sincronización. En cuanto a su eficiencia, queda condicionada por algunos factores inherentes a la técnica de reconciliación de conjuntos y otros factores condicionales relacionados con la estructura y contenido de la base de datos, tales factores pueden ser controlados de manera que los problemas de ineficiencia sean reducidos aprovechando al máximo los recursos de hardware disponible.

De esta manera se concluye el presente trabajo en el cual los objetivos planteados se han alcanzado e incluso algunos objetivos y descubrimientos que no se contemplaron al inicio.

Bibliografía

[1] Number theory library, version 5.5. http://www.shoup.net/ntl/.

[2] Claudia Morales Almonte. Instrumentación y Evaluación de Algoritmos Distribuidos Eficientes para Medir Inconsistencias entre Réplicas en Bases de Datos Relacionales Distribuidas. Master's thesis, UNAM, 2008.

[3] M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM, vol. 42, no. 1, pp. 269–291*, 1995.

[4] Edgar Frank Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM. Vol. 13. n.º 6. pp. 377–387.*, 1970.

[5] CPISync. Boston university, laboratory of networking and information systems. http://ipsit.bu.edu/nislab/projects/cpisync/download.htm, 2001.

[6] Javier García-García and Carlos Ordonez. Consistency-aware evaluation of olap queries in replicated data warehouses. In *DOLAP '09: Proceeding of the ACM twelfth international workshop on Data warehousing and OLAP*, pages 73–80, New York, NY, USA, 2009. ACM.

[7] R.J. Lipton. Efficient checking of computations. *STACS, pp. 207–215*, 1990.

[8] Yaron Minsky, Ari Trachtenberg, and Richard Zippel. Set Reconciliation with Nearly Optimal Communication Complexity. *IEEE Transaction on Information Theory*, 2004.

[9] M. Tamer Ozsu and Patrick Valduriez. *Principles of Distributed Database Systems*. 1999.

[10] PostgreSQL8.4. Postgresql global development group. http://www.postgresql.org/.

[11] Abraham Silberschatz and S. Sudarshan Henry Korth. *Fundamentos de Bases de Datos, 4ta Edicion*. 2002.

[12] Benchmark TPC-H. Transaction processing performance council. http://www.tpc.org/tpch/default.asp, 2005.