UNIVERSIDAD DE LA SALLE

FACULTAD DE CIENCIAS DE LA INGENIERÍA

CARRERA DE INGENIERÍA DE SISTEMAS

REPLICACIÓN ASÍNCRONA DE BASES DE DATOS UTILIZANDO ALGORITMOS EVOLUTIVOS

Por :

Mijael Colquehuanca Javieri

Tutor: Ing. Juan Pablo Von Landwüst

Tesis de grado presentada para la obtención del Grado de Licenciatura en Ingeniería de Sistemas

La Paz, Bolivia

Abril,2015

Dedico esta tesis:

A la memoria de mi querido abuelo, Feliciano Colquehuanca (+), que Dios y la Virgen, lo tengan en su gloria.

AGRADECIMIENTOS:

A mis padres (Justo Colquehuanca y Juana Javieri), y a mis hermanas (Alejandra y Nicole), por el gran apoyo incondicional espiritual.

A Juan Pablo, Paola (docentes), pilares fundamentales que guiaron el rumbo exacto de esta investigación.

A mis futuros colegas (compañeros de curso), docentes, gracias por todo lo vivido, y por lo que nos falta por vivir.

Gracias, a cada uno de ustedes, por ser siempre parte importante en mi vida.

"¿Normal? ¿Qué es normal? En mi opinión, lo normal es sólo lo ordinario, lo mediocre, LO ABURRIDO. La vida pertenece a aquellos individuos raros y excepcionales que se atreven a ser DIFERENTES. "

Fragmento del libro "Mi dulce Audrina"

ANDREWS V. 2006

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Las bases de datos, juegan un papel muy importante en el mundo de los negocios, a través de ellas, las empresas obtienen información que les permite tomar decisiones, sobre el lanzamiento, distribución y elaboración de un nuevo producto o servicio.

Actualmente la cantidad de información que se genera a nivel mundial cada año, se duplica, y a finales de 2011, se alcanzó almacenar, la cantidad de 1.8 zettabytes, es decir, 1.8 trillones de gigabytes (gb).

Esto quiere decir, que ante esta creciente demanda de información, es necesario elaborar tecnologías que permitan analizar las bases de datos al momento y de manera adecuada, al fin de estar al día y dar soluciones rápidas.

Una de las técnicas actuales de optimización de información en una base de datos, es la replicación, que es el proceso de copiar y mantener objetos de base de datos en varias bases de datos que componen un sistema de base de datos distribuida.

Los cambios aplicados a un sitio se capturan y se almacenan localmente antes de ser transmitidos y aplicados a cada uno de los lugares remotos.

La replicación proporciona al usuario un acceso rápido y local para los datos compartidos, y protege la disponibilidad de aplicaciones; incluso si un sitio no está disponible, los usuarios pueden consultar o incluso actualizar las ubicaciones restantes.

Para minimizar estas dificultades, la presente tesis trata de contribuir a los procesos de optimización de manejo y administración de bases de datos distribuidas, y a partir de ella, mejorar la lógica de transferencia de datos, en procesos de replicación de bases de datos tradicional, a través del estudio de algoritmos del tipo evolutivo asíncrono, basado en heurísticas y modelos formales orientados a la lógica matemática, codificado en pseudocódigo y testeado en bases de datos grandes.

El presente documento está dividido en cinco capítulos y dos anexos.

El capítulo 1, presenta, todo lo referente a la parte introductoria, y describe entre otros, la importancia y los objetivos que se pretende alcanzar en el desarrollo de esta tesis de grado.

El capítulo 2, presenta todos los fundamentos teórico/prácticos que fueron necesarios para el desarrollo de la presente investigación.

El capítulo 3, presenta una estructura algorítmica del tipo evolutivo asíncrono, codificado en pseudocódigo, testeado en bases de datos transaccionales grandes, este como propuesta de solución a la hipótesis planteada en la presente investigación.

El capítulo 4, presenta los resultados finales de la presente investigación, a través del estudio de variables y el análisis estadístico, en comparativa a otras estructuras similares, de cada una de las pruebas ejecutadas en bases de datos transaccionales grandes.

El capítulo 5, presenta las conclusiones y recomendaciones para futuros trabajos de investigación, acerca de este campo.

Finalmente se anexan al presente trabajo, tres documentos: la configuración del Sistema Gestor de bases de datos, la guía de usuario para la ejecución correcta de la estructura algorítmica y un código de apoyo propio para el análisis estadístico de variables .

CAPÍTULO I

MARCO REFERENCIAL

1.1 Introducción

Hoy en día , en nuestro diario vivir, vamos generando volúmenes grandes de información , la misma, que necesita ser almacenada en forma ordenada y segura , en repositorios de datos, banco de datos o más conocidas como bases de datos.

Las bases de datos, juegan un papel muy importante en el mundo de los negocios, a través de ellas, las empresas obtienen información que les permite tomar decisiones, sobre el lanzamiento, distribución y elaboración de su muevo producto o servicio.

La importancia de las bases de datos radica principalmente, en que ayudan a las empresas en la toma de decisiones, gestionar opiniones de los clientes, sacar al mercado un producto o generar una campaña de marketing cuando se requiere.

La cantidad de información que se genera a nivel mundial cada año, se duplica, y a finales de 2011, se alcanzó almacenar, la cantidad de 1.8 zettabytes1, es decir, 1.8 trillones de gigabytes [BNY 13].

Antes esta creciente avalancha de datos, la industria de almacenamiento digital enfrenta el reto de ofrecer soluciones que permitan administrar y almacenar grandes cantidades de información en menos espacio, de una forma más rápida, segura y sustentable.

1.2 Antecedentes

A continuación detallaremos, una serie de trabajos previos, realizados en torno a la presente investigación:

1.2.1 Protocolos de replicación de bases de datos

José Ramón Juárez Rodríguez, ingeniero de telecomunicación, ha investigado en su tesis doctoral [JUA 11] cómo mejorar los protocolos de replicación de bases de datos. En su tesis ha estudiado como son los procesos de transmisión de información entre diferentes réplicas, de tal forma que la información transmitida afecte o actualice a cada uno de los nodos incluidos.

[1] Un zettabyte es una unidad de almacenamiento de información cuyo símbolo es el ZB, equivale a 1021 bytes.

1.2.2 Replicación Autonómica de Bases de Datos

Jesús Manuel Milán Franco, estudió la adaptabilidad dinámica en el contexto de las bases de datos replicadas basadas en middleware[2]. El trabajo realizado en esta tesis [MIL 08] desarrolla un sistema de replicación adaptable a nivel de middleware que englobe ambas áreas de trabajo desarrollando protocolos y algoritmos para el control de la concurrencia a nivel local y el equilibrado de carga a nivel global y que maximizan de forma automática el rendimiento del sistema bajo distintos tipos de carga, fallos y recuperación de réplicas.

1.2.3 Lenguaje XML como solución a las bases de datos y su replicación

René Amílcar Monroy Hernández, en su tesis doctoral [MON 05] propone, demostrar la forma de como los diferentes proveedores de bases de datos aplican y utilizan soporte XML[3] en bases de datos para su replicación.

1.2.4 Análisis de rendimiento y replicación en Bases de Datos distribuidas

Rodolfo Bertoné , en su artículo denominado : Análisis de rendimiento y replicación en Bases de Datos distribuidas [BER 11], escribió acerca del estudio del comportamiento (en tiempo de respuesta y confiabilidad) de grandes bases de datos de imágenes sobre arquitecturas distribuidas de redes LAN y WAN.

Esto significa problemas tales como: Tasa de pérdida de datos; Tiempo máximo necesario para recuperación de información; Complejidad y eficiencia de los algoritmos de recuperación;

Tiempo de utilización de recursos del sistema; Incidencia del porcentaje de replicación en el tiempo de respuesta.

1.3 Planteamiento del problema

Grandes organizaciones, tales como bancos o Administraciones Públicas, supermercados, hospitales, hoteles, etc ; no pueden permitirse la pérdida de información, ni el cese de operaciones ante un desastre en su centro de proceso de datos. Terremotos, incendios o atentados en estas instalaciones son infrecuentes, pero no improbables. Por este motivo, se suele habilitar un centro de respaldo para absorber las operaciones del CPD (Central de

[2] Middleware es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos. Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones que son necesarias en los sistemas distribuidos.

[3] XML, siglas en inglés de eXtensible Markup Language ('lenguaje de marcas extensible'), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.

procesamiento de datos) principal en caso de emergencia, a estos CPD´S se los conoce con el nombre de backups, copias de respaldo o más conocidas como métodos de replicación de base de datos.

Algunos años atrás, los datos más importantes de las empresas, se encontraban almacenados en un repositorio central. Departamentos remotos accedían a la información que necesitaban por medio de conexiones directas al repositorio central, o por solicitudes del sistema de reportes impresos desde el Sistema de Información Gerencial. Sin embargo, las conexiones eran costosas, poco fiables y limitadas en el número de usuarios concurrentes, al mismo tiempo los reportes eran poco flexibles y su generación era lenta.

Luego aparecieron sistemas abiertos de bajo costo en recursos de computación y de gran alcance a los diferentes departamentos remotos de la empresa. La posibilidad de compartir la base de datos de la empresa efectivamente y la utilización eficaz de los recursos se convirtieron en ventajas competitivas para las organizaciones. Hoy en día las empresas no se preguntan "¿Por qué distribuir y compartir los datos de la organización?" sino más bien, "¿Cómo se puede distribuir información de manera efectiva?". Hoy en día, la replicación se está convirtiendo rápidamente en la mejor elección de arquitectura de aplicaciones distribuidas para la mayoría de las empresas. .

Por lo que ante el mundo cambiante es necesario elaborar tecnologías que permitan analizar las bases de datos al momento y de manera adecuada, al fin de estar al día y dar soluciones rápidas.

Ninguna empresa existiría si no tuviera clientes que atender, por ello, miles de empresas en el mundo dedican gran parte de su tiempo y esfuerzo a tratar de incrementar el número de retención de datos de los clientes, y de acuerdo a cuanto sabemos del cliente, podemos medir su grado de satisfacción.

Ayudan en el desarrollo de modelos de administración, que permiten capturar y analizar sistemáticamente, la información proveniente de los clientes, con la finalidad de captar las diferencias, por más pequeñas que sean, entre estas.

Realizado un análisis previo entorno a este tema, se vio que los procesos actuales de distribución y acceso remoto a bases de datos transaccionales grandes ya mencionados, eran demasiado lentos, y poco fiables, esto debido a que la afluencia de información era demasiado alta, y en el desarrollo de los procesos ocasionarían daños a la base de datos, ya

sea que una actualización se haya o no realizado en un momento determinado, o tal vez que alguna información se haya perdido durante el proceso.

Una de las alternativas de solución, fue replicar la base de datos principal, a otros repositorios secundarios, pero de igual forma los tiempos en procesos de replicación de información eran altos en cuanto las bases de datos iban creciendo cada vez más.

Entonces a partir de varias investigaciones realizadas acerca de esta problemática, pudimos deducir que la lentitud en los procesos de replicación se debe, a que los métodos de replicación (algoritmos basados en modelos de replicación síncrona) utilizados actualmente, no eran los adecuados cuando la cantidad de datos va en aumento.

1.4 Formulación del problema

¿Cómo es posible mejorar la lógica de transferencia de datos, en los procesos de replicación de bases de datos?

1.5 Objetivos

1.5.1 Objetivo general

Mejorar la lógica de transferencia de datos, en procesos de replicación de bases de datos tradicionales, a través del estudio de algoritmos del tipo evolutivo asíncrono, basado en heurísticas y modelos formales orientados a la lógica matemática, codificado en pseudocódigo y testeado en bases de datos transaccionales grandes, ayudando así en procesos de replicación de bases de datos.

1.5.2 Objetivos específicos

- Analizar los métodos y técnicas de elaboración de algoritmos evolutivos, del tipo asíncrono, basados en heurísticas y modelos formales orientados a la lógica matemática, representados en pseudocódigo para la implementación del mismo.

- Analizar técnicas de replicación de bases de datos tradicionales, para determinar los puntos críticos, en el desarrollo del proceso.

- Definir una estructura algorítmica basada en heurísticas matemáticas, que nos permitan reducir los tiempos de transferencia de datos, en el proceso de la replicación.

- Desarrollar el prototipo para demostrar los resultados del funcionamiento de la estructura algorítmica.

1.6 Hipótesis

La aplicación de algoritmos evolutivos asíncronos, ayuda a mejorar la lógica de transferencia de datos en los procesos de replicación de bases de datos transaccionales grandes.

1.7 Justificaciones

1.7.1 Científica

La presente investigación pretende brindar metodológicamente un esquema algorítmico del tipo evolutivo asíncrono, basado en heurísticas y modelos formales orientados a la lógica matemática, aplicable durante los procesos de replicación de datos en bases de datos transaccionales de gran tamaño; por lo tanto, al plantearse esta investigación, sin duda, aumentara el conocimiento en el área de estudio, dando lineamientos básicos para continuar con investigaciones futuras, relacionadas al tema.

1.7.2 Tecnológica

Con el avance de la tecnología y producto de ella, el incremento de la información, que día a día va creciendo exponencialmente a la par de la tecnología de almacenamiento físico, son más necesarios los estudios de métodos y técnicas de replicación de bases de datos.

1.7.3 Económica

Toda investigación que coadyuve en mejorar los procesos de replicación de información de bases de datos, no significa en definitiva un gasto, en relación a los beneficios que pueden obtenerse a través de la aplicación correcta de la estructura algorítmica basada en heurísticas y modelos formales orientados a la lógica matemática.

1.7.4 Social

La presente investigación , no solo tiene justificaciones del tipo tecnológico o científico, ya que, al proponer un algoritmo, que coadyuve en disminuir los tiempos de replicación de bases de datos, consecuentemente los tiempos de mantenimiento por replicación de información en las empresas serán más cortos, lo que en pocas palabras quiere decir para el cliente : menos caídas de sistema, menos cierres de entidades por tema de mantenimiento de replicación de información.

1.8 Limites y aportes

1.8.1 Limites

Debido a la complejidad del tema, y al tiempo asignado para la presente investigación; como producto final se presentará un prototipo, que básicamente costa de: una estructura algorítmica del tipo evolutivo asíncrono, codificado en pseudocódigo, testeado en bases de datos transaccionales grandes, y analizado estadísticamente en comparativa a otros similares.

1.8.2 Aportes

Mejorar la lógica de replicación de bases de datos transaccionales tradicionales, mediante el uso de algoritmos evolutivos asíncronos, logrando así, la mejora en tiempos de transferencia de datos de replicación.

1.9 Modelos y herramientas

1.9.1 Herramientas técnicas de análisis y diseño

- Técnica de investigación documental.

- Técnica de diseño de algoritmos.

- Herramientas CASE y CAD.

1.9.2 Herramientas técnicas de implementación

- Lenguaje de programación Python, Perl, Javascript y PHP.

- Lenguaje de consultas de bases de datos SQL.

- Lenguaje de procedimientos y consultas estructuradas PLSQL.

- Sistema gestor de bases de datos Postgres SQL.

- Sistema manejador de base de datos PGADMIN3.

- Herramienta de replicación de datos BUCARDO.

1.10 Metodología del proyecto

- Método científico de investigación.

- Método de ordenación y búsqueda de algoritmos.

- Métodos de resolución de algoritmos en base a heurísticas.

- Métodos recursivos de resolución de algoritmos.

- Métodos de programación orientados a objetos.

- Método de resolución de problemas algorítmicos “Back tracking” (en castellano “Dando marcha atrás”).

- Método de resolución de problemas algorítmicos “Divide and rule”(en castellano “Divide y vencerás”).

- Método de sincronización de bases de datos “On Server Startup” (en castellano “En el inicio del servidor”).

- Método de replicación asíncrona de base de datos.

- Otros métodos de replicación síncrona de base de datos.

CAPÍTULO 2

MARCO TEÓRICO

El presente capítulo muestra una descripción general de toda la base teórica práctica utilizada para la realización de la investigación; dando a conocer lineamientos básicos de replicación, bases de datos y algoritmia; así como también las estructuras algorítmicas testeadas, basados en modelos evolutivos de programación.

2.1 Replicación de bases de datos

La replicación de bases de datos es la creación y el mantenimiento de múltiples copias de la misma base de datos. En la mayoría de las implementaciones de replicación, un servidor mantiene la copia maestra de la base de datos y los servidores adicionales mantienen copias esclavo.

Hay muchas técnicas para gestionar la replicación. Éstas se pueden clasificar de diferentes maneras. En este módulo nos fijaremos en dos parámetros para presentar dos posibles clasificaciones:

1) qué réplica se cambia y

2) cuándo se propagan las modificaciones al resto de réplicas.

Según el primer parámetro, los protocolos de replicación se pueden clasificar en single-master y multi-master. Según el segundo parámetro, en síncronas (eager) o asíncronas (lazy).

2.1.1 Single frente a multi-masters

En el caso del single-master hay una copia principal de cada objeto, que la llamaremos primaria. Cuando hay una modificación, ésta se aplica primero a la copia primaria. Después se propaga al resto de copias (que son las secundarias). En este modelo se puede leer cualquier réplica de un objeto, pero sólo se puede modificar la primaria.

En la aproximación multi-master hay varios almacenes que contienen una copia primaria de un mismo objeto. Todas estas copias se pueden actualizar de forma concurrente. En este caso se puede acceder por lectura a cualquiera de las copias y por modificación a cualquiera de las primarias.

La aproximación multi-master reduce los cuellos de botella y los puntos de fallo, así también existen protocolos que aprovechan las ventajas de los sistemas de comunicación en grupo para evitar así algunos problemas de rendimiento.

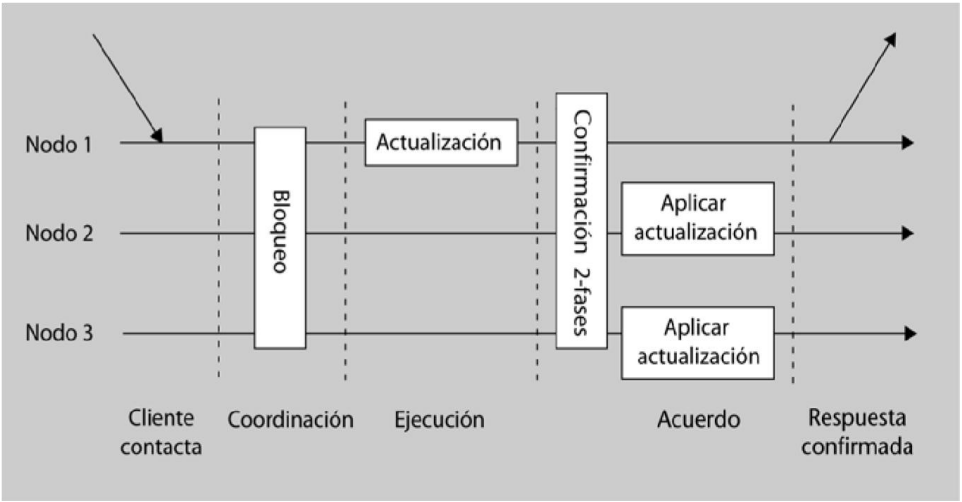
La figura 1 muestra las diferentes etapas que sigue un sistema distribuido síncrono que utiliza un algoritmo de confirmación en dos fases para actualizar un objeto. La operación de actualización se inicia en el nodo 1. Como parte de ésta, se bloquean todas las réplicas del dato. Seguidamente se hace la modificación en el nodo 1 y, utilizando un algoritmo de confirmación en dos fases, la actualización se propaga al resto de nodos.

Finalmente, la operación de actualización acaba. En este momento todas las réplicas del objeto tienen el mismo valor. Es importante destacar que la operación no retorna hasta que todas las réplicas han aplicado la actualización.

Figura 1

Etapas que sigue un sistema distribuido síncrono.

Fuente : [STA 13]



Con este tipo de técnicas se consigue que, aunque haya varias copias de un mismo objeto, el usuario perciba que el comportamiento es como si sólo hubiera una. Este criterio de consistencia se conoce como one-copy seriability.

La gran ventaja de los protocolos síncronos es que evitan la divergencia entre réplicas de un mismo dato.

El gran inconveniente es que cualquier escritura tiene que actualizar muchas o todas las réplicas antes de finalizar. Eso es un inconveniente para sistemas cuyos nodos sean dinámicos (sistemas de igual a igual o sistemas que permiten el trabajo en desconectado) o en entornos de gran alcance donde a causa de la latencia, es costoso en tiempo actualizar todas las réplicas.

Además, tiene grandes limitaciones de escalabilidad debidas al tiempo necesario para actualizar todas las réplicas.

En los sistemas asíncronos no hace falta que se actualicen todas las copias de un objeto como parte de la operación que inicia la actualización. En este tipo de sistemas la operación de actualización acaba en cuanto puede, y posteriormente el cambio se hace llegar al resto de réplicas.

2.2 Evaluando algoritmos de replicación síncrona

2.2.1 Reserva en dos fases

La reserva en dos fases es un mecanismo sencillo para garantizar seriabilidad y la primera copia seria (en inglés, one-copy seriability).

Seriabilidad: propiedad que provoca que el resultado de ejecutar una operación sea el mismo que si el resultado se hubiera ejecutado de manera secuencial (sin superposiciones debidas a la concurrencia).

One-copy seriability: propiedad que genera que un usuario perciba el comportamiento de un conjunto de copias de un dato replicado como si sólo hubiera uno.

El protocolo de reserva en dos fases gestiona las reservas (locks, en inglés) durante la ejecución de la transacción en las dos fases siguientes:

- Se adquieren todas las reservas y no se libera ninguna.
- Se liberan las reservas y no se adquiere ninguna.

Se garantiza la seriabilidad para ejecuciones que sigan este orden en la gestión de las reservas.

2.2.2 Confirmación distribuida

Los algoritmos de confirmación distribuida son útiles para situaciones en las que interesa garantizar que todos los procesos de un grupo ejecutan una operación o que ninguno de ellos la ejecuta. En el caso de multicast fiable, la operación que se ejecuta sería la entrega del mensaje. En el caso de transacciones distribuidas, la operación sería la realización de la transacción.

Generalmente, las operaciones de confirmación distribuida se basan en un coordinador que notifica al resto de procesos que ya pueden realizar (o no) la operación en cuestión (en local). Claramente ya se ve que si uno de los procesos no puede hacer la operación, no hay manera de notificarlo al coordinador.

Por este motivo son necesarios unos mecanismos más sofisticados para poder hacer estas confirmaciones distribuidas. A continuación presentamos la confirmación en dos fases y la confirmación en tres fases.

2.2.3 Confirmación en dos fases

Es un algoritmo muy popular para hacer la confirmación distribuida. Se basa en tener un coordinador y el resto de procesos. Si suponemos que no hay fallos, el funcionamiento del algoritmo sería el siguiente:

2.2.3.1 Fase petición de confirmación

El coordinador envía una petición de confirmación al resto de participantes.

El coordinador espera hasta que recibe un mensaje de cada uno del resto de participantes.

Cuando un participante recibe un mensaje de petición de confirmación contesta al coordinador un mensaje indicando si está de acuerdo en hacer la confirmación local o de aborto si no la puede hacer.

2.2.3.2 Fase de confirmación

2.2.3.2.1 Éxito

Si el coordinador ha recibido un mensaje de todos los participantes en la fase de petición de confirmación con la indicación de que están de acuerdo en hacer la confirmación:

- El coordinador envía un mensaje de confirmación a todos los participantes.

- Cada participante completa la transacción, y libera todos los recursos y las reservas adquiridas durante la transacción.
- Cada participante envía un mensaje de acuse de recibo (acknowledgement, en inglés) al coordinador.
- El coordinador acaba la transacción una vez ha recibido todos los acuses de recibo.

2.2.3.2.2 Fracaso

Si alguno de los participantes contesta un mensaje de aborto durante la fase de petición de confirmación:

El coordinador envía un mensaje a todos los participantes para que deshagan las operaciones que hayan podido realizar o liberen los recursos o las reservas que tengan ocupadas.

Cada participante deshace las posibles operaciones que ya haya hecho y libera las reservas y los recursos que tenga ocupados.

Cada participante envía un mensaje de acuse de recibo al coordinador.

El coordinador acaba la transacción una vez ha recibido todos los mensajes de acuse de recibo.

La gran desventaja del protocolo de confirmación en dos fases es el hecho de que es un protocolo que bloquea. Un participante se bloqueará mientras espere un mensaje.

Eso quiere decir que otros procesos que estén compitiendo por la reserva de un recurso que tiene ocupado el proceso bloqueado deberán esperar a que éste libere la reserva. Un proceso continuará esperando incluso cuando el resto de procesos han fallado. Si el coordinador falla de manera permanente, algunos participantes no resolverán nunca la transacción. Esto provoca que los recursos estén ocupados para siempre.

El algoritmo se puede bloquear indefinidamente en los casos siguientes: cuando un participante envía un mensaje de acuse de recibo al coordinador hace que se quede bloqueado hasta que recibe un mensaje de confirmación o de deshacer las operaciones realizadas.

Si el coordinador falla y no se vuelve a recuperar, los participantes se quedarán bloqueados hasta que el coordinador se recupere o, en el peor de los casos, indefinidamente, ya que éste no puede decidir por su cuenta abortar o confirmar. Lo único que se podría intentar es averiguar qué mensaje había enviado al coordinador.

Por otro lado, el coordinador se quedará bloqueado una vez haya hecho la petición de confirmación hasta que todos los participantes le contesten. Si un participante está indefinidamente parado, el coordinador decidirá abortar cuando salte el temporizador del nodo que está parado. Esta decisión también es una desventaja del protocolo porque está sesgado hacia el aborto en lugar de estarlo hacia la finalización.

2.2.4 Confirmación en tres fases

Un problema del protocolo de confirmación en dos fases es que cuando el coordinador falla, los participantes pueden no ser capaces de llegar a una decisión final. Eso puede provocar que los participantes se queden bloqueados hasta que el coordinador se recupere. Aunque el protocolo de confirmación en tres fases sea no bloqueante, no se utiliza demasiado en la práctica, ya que las condiciones en las que el protocolo de confirmación en dos fases se bloquea ocurren raramente.

Más concretamente, el protocolo de confirmación en tres fases fija un umbral superior en la cantidad de tiempo necesario antes de que una transacción o bien confirme o bien aborte. Esta propiedad asegura que si una transacción intenta confirmar vía el protocolo de confirmación en tres fases y toma alguna reserva de un recurso, liberará la reserva después de expirar un temporizador asociado.

2.2.4.1 Coordinador

El coordinador recibe una petición de transacción. Si hay algún fallo en este punto, el coordinador aborta la transacción (es decir, cuando se recupere, considerará la transacción abortada). De lo contrario, el coordinador envía un mensaje de inicio de transacción a los participantes y se pone en estado de espera.

Si hay un fallo, expira el temporizador, o si el coordinador recibe un mensaje de algún participante que avisa de que no está en disposición de iniciar la transacción durante la fase de espera, el coordinador aborta la transacción y envía un mensaje de aborto a todos a los participantes. De lo contrario, el coordinador recibirá mensajes de aceptación del inicio de la

transacción de los participantes dentro de la ventana de tiempo, y enviará mensajes de confirmación a todos los participantes. A continuación, se pone en estado preparado.

Si el coordinador falla en el estado de preparado, se moverá al estado de confirmación. En cambio, si el temporizador del coordinador expira mientras está esperando la confirmación de uno de los participantes, abortará la transacción. En caso de que se reciban todos los acuses de recibo, el coordinador también cambia al estado de confirmación.

2.2.4.2 Participante

El participante recibe un mensaje de inicio de transacción del coordinador. Si el participante está de acuerdo, contesta con un mensaje indicando que está en disposición de iniciar la transacción y se cambia al estado preparado. De lo contrario, envía un mensaje en el que indica que no está en disposición de iniciar la transacción y aborta. Si hay un fallo, se mueve al estado de abortar.

En el estado de preparado, si el participante recibe un mensaje de aborto del coordinador, falla, o si expira el tiempo de espera para una confirmación, aborta. Si el participante recibe un mensaje de confirmación, contesta un mensaje de acuse de recibo y confirma.

La principal desventaja de este algoritmo es que no se puede recuperar de un fallo de partición de la red. Es decir, si los nodos se separan en dos mitades iguales, cada mitad continuará por su cuenta.

2.3 Manos a la obra

Debido a las limitaciones, que nos proporcionan la utilización de métodos y algoritmos del tipo síncrono, ya mencionados anteriormente, esta tesis propone una alternativa de solución en base a la organización y reestructuración de los datos a replicar, con el fin de poder mantener actualizados los viejos y nuevos datos en las base de datos, proponiendo básicamente tres etapas: recolección de información de la transacción, mapeamiento y la ejecución.

2.3.1 Recolección de información de la transacción

La etapa de recolección de información de las transacciones es ejecutada a partir de un trigger bastante simple y genérico, al cual se lo denominó trigger recolector, todo esto, con el objetivo de recolectar toda la información de la transacción, por ejemplo referente a si ha sido

insertada, actualizada, o borrada o saber cuáles fueron los valores anteriores o cuales son los valores que serán actualizados.

En esta etapa básicamente, se realizara lo que comúnmente se llama una ¨Normalización de la base de datos¨, a partir de un trigger recolector.

Para ejemplificar el presente capítulo utilizaremos, una base de datos auxiliar, denominada CONTABLE, descrita en el anexo D, del presente documento.

La figura 2 describe un típico ejemplo de recolección de información, basado en la estructura propuesta; en este caso de las tablas detalle y entidad.

Figura 2

Información de las transacciones

Fuente: Elaboración propia.

La figura 3 muestra dos tablas temporales, utilizados por los triggers recolectores para el almacenamiento de las transacciones. Se estructuran de la siguiente manera:

- Tres columnas de control: Se identifican tres columnas de control, la primera la clave primaria de la tabla temporal, la segunda el estado de replicación, y la tercera a la operación que se ha efectuado (inserción, exclusión, actualización).

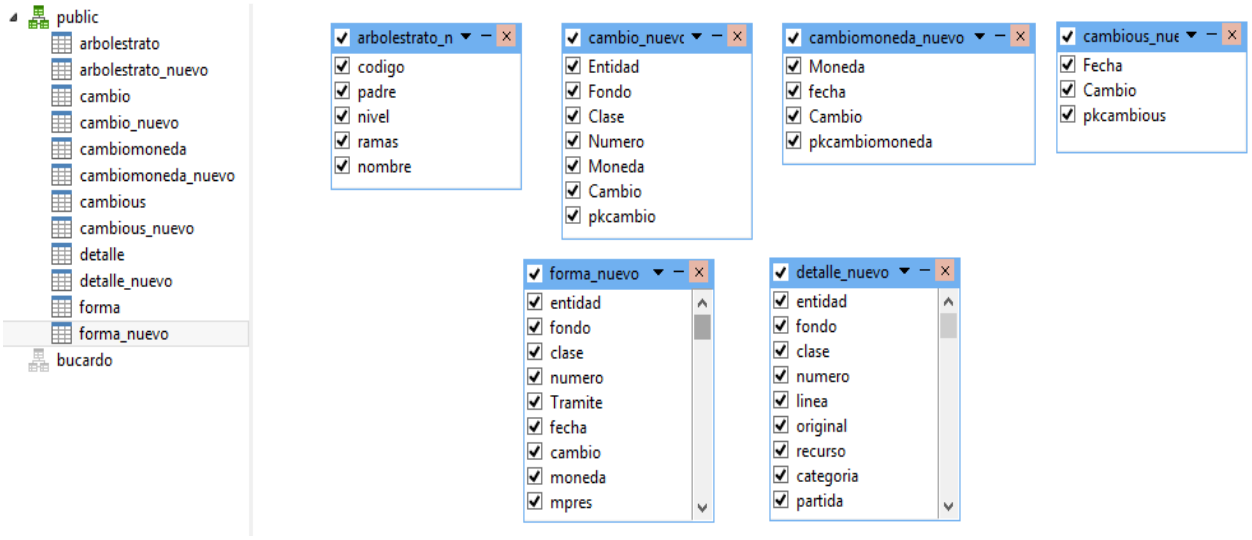
- Tabla de origen de llaves. Identificación de las claves primarias, tanto en el origen, como en el temporal.

- Columnas viejo y lo nuevo: Para cada columna de la tabla de origen, hay dos columnas, una con el prefijo de NEW (en castellano nuevo) y otra con el prefijo OLD (en castellano viejo), para almacenar el valor de la columna antes y después de la transacción.

Figura 3

Tablas temporales

Fuente: Elaboración propia.



2.3.2 Etapa de mapeamiento

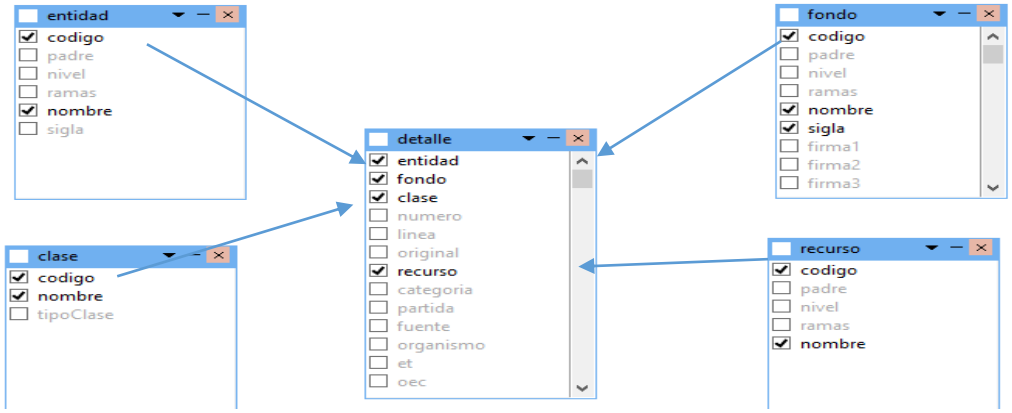
El objetivo de la etapa de mapeo es definir un mapa del origen al destino de las transacciones.

Por ejemplo en la figura 4 , podemos ver que la tabla principal llamada detalle, tiene como origen los campos entidad, fondo, clase, recurso ; y como destino las tablas entidad, fondo , clase y recurso; obviamente cada una con sus respectiva clave primaria , antecesora a su clave foránea.

Figura 4

Mapeo de la base de datos

Fuente: Elaboración propia.



Por lo tanto, para este ejemplo, estructuradamente nuestro mapeamiento de la base de datos, sería el siguiente:

Figura 5

Mapeamiento de la tabla detalle

Fuente: Elaboración propia.

mapeo_detalle
detalle_entidad =>entidad_codigo=>if(codigo=true)(entidad=>nombre)
detalle_fondo =>fondo_codigo=>if(codigo=true)(entidad=>nombre)
detalle_clase =>clase_codigo=>if(codigo=true)(entidad=>nombre)
detalle_recurso =>recurso_codigo=>if(código=true)(entidad=>nombre)
…… y así sucesivamente para cada relación.

Aplicamos el mismo procedimiento, para cada campo relacional de la tabla detalle, como vemos en la figura anterior.

Obviamente realizamos el mismo paso, para cada tabla de la base de datos.

2.3.3 Etapa de ejecución

Con cierta frecuencia, el administrador de base de datos, realiza el proceso de replicación de los datos, todo esto con el fin de que los esquemas viejos y nuevos se mantengan actualizados.

El objetivo es procesar la información recogida en la primera etapa, siguiendo las asignaciones de la segunda.

La figura 6 muestra un proceso de replicación en funcionamiento. A continuación, se puede ver las principales acciones llevadas a cabo:

- Mapeo de lectura: El proceso lee el mapa para deducir el origen y destino de la información.

- Uso de transacciones recogidas: La información de las transacciones se leen y se procesan de manera que es posible reproducir las operaciones en la tabla de destino.

- Cambiadores: Las tablas definidas geográficamente conforme al proceso de mapeamiento se actualizan.

- Grabación de registros de ejecución: El resultado se registra en cada tabla y campo afectado.

Cuando los tres pasos se encuentran dentro de los disparadores en el formato propuesto por Ambler [AMB 06], la actualización de los esquemas es sincrónica.

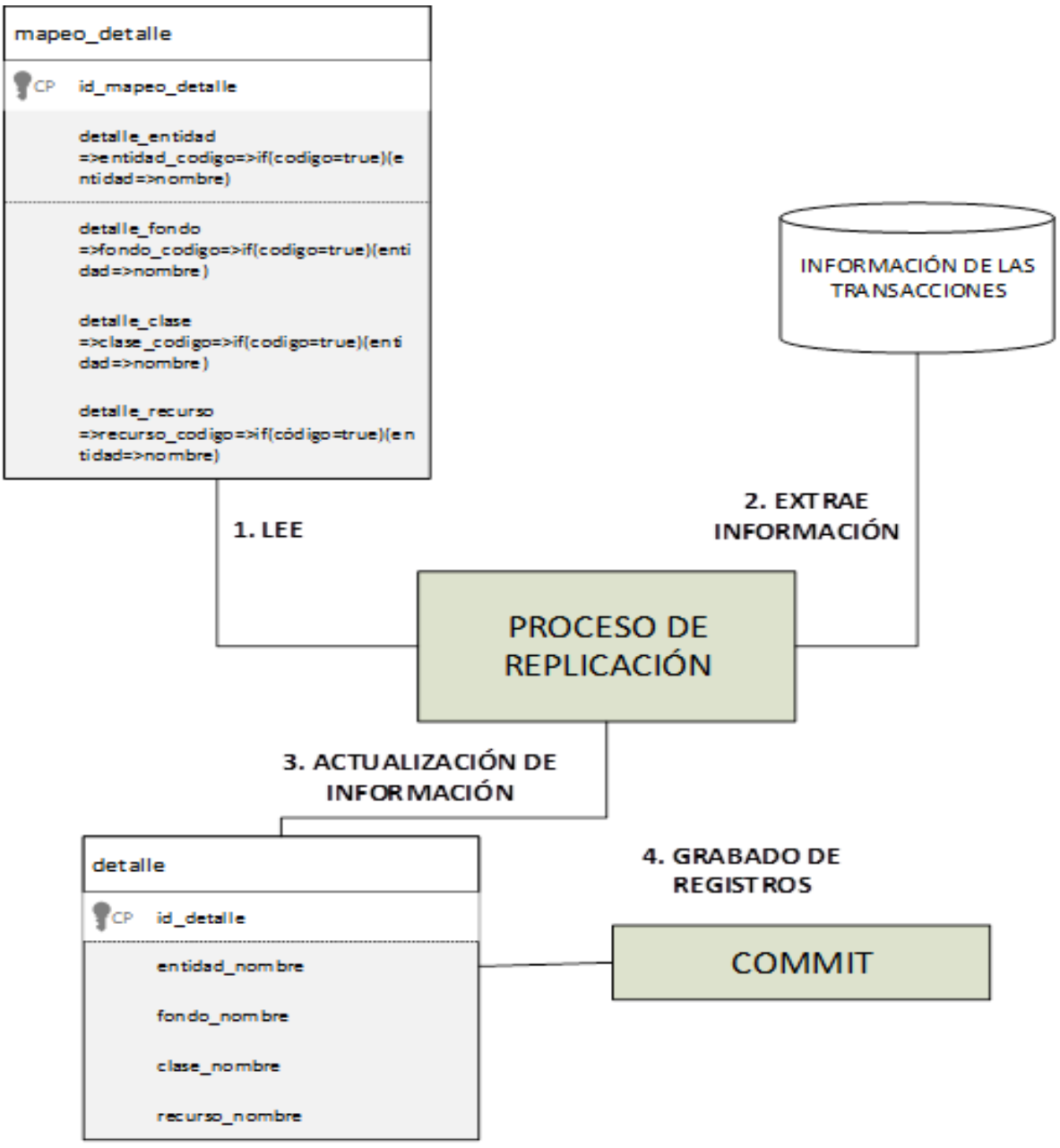
A diferencia, que cuando sólo la etapa de agrupamiento o recolección se realiza junto con la transacción de aplicación y la fase de ejecución (que es el paso que en realidad realiza el trabajo de actualización) se lleva a cabo en un momento posterior, se puede llamar a la actualización de forma asíncrona.

Como veremos, en la presente tesis, propone una nueva forma de estructuración del algoritmo, que básicamente permite la resolución de los problemas señalados en el enfoque propuesto por Ambler [AMB 06].

Figura 6

Proceso de replicación en general.

Fuente: Elaboración propia.



2.3.4 Evaluación preliminar al trabajar bajo la replicación asincrónica

Basados en el análisis de las dificultades que se presentan en el capítulo 2, tenemos la siguiente evaluación preliminar de la replicación asincrónica:

- Codificación específica: no vamos a tener un disparador específico para cada tipo de refactorización. Básicamente solo necesitaremos un disparador genérico que almacena toda la información existente en una transacción.

- Código de ciclos de tratamiento: El problema se resolverá en un solo punto: en la fase de aplicación. El proceso de replicación tiene acceso a todas las asignaciones existentes y también a las transacciones captadas hasta el momento. Esto hace que sea posible deducir la existencia de ciclos y evitar el problema.

- Transacción lenta: una aplicación de transacción no será lento, porque el único disparador que será ejecutado será el colector, que es bastante simple y se realiza únicamente al almacenamiento de información en una transacción.

- Posibles errores en la aplicación: La probabilidad de error es menor al 50% ya que los procesos ejecutados, van a almacenar todos los registros necesarios, alertando responsables y en caso de ocurrencia , se procederá a la realización de la réplica de nuevo cuando el problema se haya resuelto.

Aparte de las dificultades que la replicación asincrónica resuelve, se obtiene un gran beneficio para dividir el proceso en tres etapas: la primera que nos da la posibilidad de desarrollar una herramienta para los administradores de bases de datos para realizar refactorizaciones.

Este proceso es importante, ya que necesitamos formatear nuestra base de datos, aun formato específico para proceder con la recolección de transacciones (primera etapa) y para ello es factible construir un disparador embebido en un aplicativo, para definir las asignaciones (segunda etapa).

Además, la herramienta puede incluir un conjunto predefinido de asignaciones para cada tipo de refactorización.

Para terminar, simplemente ejecutamos un proceso que lee las asignaciones y utiliza la información almacenada de las transacciones, la actualización de los destinos de las tablas (tercera etapa), y ya con todo ello, procedemos a la replicación.

2.3.5 Implementación de un proceso de replicación

La premisa principal para la replicación asíncrona es la posibilidad de que las tablas que participan en una refactorización queden desactualizadas con los datos actuales en un determinado tiempo. Esto dependerá del período de tiempo establecido por el administrador de la base de datos, este tiempo es el intervalo entre ejecuciones del proceso de replicación.

El administrador puede decidir si el proceso debe ser realizado, por ejemplo, una vez al día. En otro, se puede definir, que el período de tiempo no sobrepase el máximo, ya que si llega a suceder, los datos se volverían obsoletos.

Por ello se recomienda que el intervalo de procesos de replicación, debe ser tan pequeño como sea posible (por ejemplo, un minuto), ya que cuanto menor tiempo estén los datos almacenados, en tablas provisorias ósea en las tablas de refactorización, menor será el espacio que utilicemos en la base de datos, y menor serán los tiempos de replicación.

La implementación de un proceso de replicación que se ejecute cada minuto de intervalo, horas o días es sencillo, sólo tiene que utilizar alguna herramienta para la planificación de los procesos de ejecución.

Si el administrador requiere un rango que es tan pequeño como sea posible, es probable que la replicación sea más sofisticada, ya que reduciremos el uso de nuestro sistema gestor de bases de datos, también reduciremos los tiempos perdidos, cuando un proceso está pendiente o está muerto, ya que previamente se realizará una evaluación, la misma que devolverá notificaciones, entorno a las dos primeras etapas del proceso y solo se replicará todos los datos que estén vivos.

Estas notificaciones, se realizarán a partir de nuestro disparador recolector, utilizando simplemente las funciones LISTEN y NOTIFY del sistema gestor de base de datos a utilizar.

En esta implementación, el período de tiempo con datos obsoletos se puede reducir a la orden de milisegundos - el tiempo entre la recepción de la notificación por el proceso dedicado y actualizar la tabla de destino.

La violación de la consistencia de los datos, aunque sea por un corto período de tiempo, se debe mostrar de forma explícita por el dominio de la aplicación para que los usuarios puedan evaluar la importancia de esa incompatibilidad.

Las instituciones financieras, por ejemplo, y siempre sabe que la posición de una cuenta corriente puede llegar a ser inconsistente hasta la finalización de todos los desplazamientos, créditos y / o descuentos.

Por otro lado, en otros ámbitos, como los sistemas de salud para las infecciones urinarias, algunas informaciones, como el último informe sobre el estado de salud del paciente, no puede ser incoherente porque los profesionales de la salud podrían realizar intervenciones incorrectas en el paciente.

En consecuencia, las consultas a los datos deben informar no sólo los datos, sino también su grado de inconsistencia, para que los usuarios puedan tomar decisiones en función de sus ámbitos de aplicación específicos.

2.3.5.1 Trigger recolector

Su estructura se define a partir de la tabla de origen de datos y contiene las siguientes columnas:

Figura 7

Estructura de un trigger recolector.

Fuente: Elaboración propia.

```
1   CREATE SEQUENCE detalle_transactions_id_fondo;
2   CREATE TABLE detalle_transactions (
3   id integer NOT NULL DEFAULT NEXTVAL('detalle_transactions_id_fondo'),
4   id_table integer NOT NULL,
5   estado character varying(50) NOT NULL,
6   operacion character varying(50) NOT NULL,
7   NEW_entidad character varying(255),
8   NEW_fondo character varying(255),
9   NEW_clase character varying(255),
10  NEW_recurso character varying(255),
11  OLD_entidad character varying(255),
12  OLD_fondo character varying(255),
13  OLD_clase character varying(255),
14  OLD_recurso character varying(255)
15  );
16  CREATE UNIQUE INDEX detalle_transactions_id ON detalle_transactions(id);
```

- Las dos primeras columnas: id es sólo una secuencia de detalle_transaction y la columna id_table es la secuencia de la tabla de cual precede.

- Estado: para todas las filas de la tabla incluyen detalle_transactions, esta columna obtiene el valor inicial de los nuevos. Después de que el proceso de replicación haga su trabajo, las filas procesadas reciben el valor de EJECUTADO . Las líneas con valor ejecutado en la columna Estado ya no se usan para las próximas ejecuciones del proceso.

- Operación de la columna:. Contiene la operación realizada en la tabla detalle. Los valores posibles son: DELETE, INSERT y UPDATE. Esta información indicará al proceso de replicación que se debe hacer en la tabla de destino.

- Columnas con nombres New + Nombre de la columna : nuevos valores de las transacciones de inclusión y actualización.

- Columnas con nombres Old + Nombre de la columna: valores de columna antes de operaciones de eliminación y actualización.

La figura 8 muestra el código del receptor de activación para la tabla detalles. Como podemos ver, el código es bastante conocido.

Hay un INSERT en la tabla temporal detalle_transactions (líneas 5, 12 y 19) para cada tipo de operación (línea 4 a DELETE, UPDATE, y la línea 11 a la línea 18 para INSERT).

Dependiendo del tipo de operación, un conjunto de valores debe ser incluido en la tabla temporal. En el caso de la exclusión, necesitamos sólo los viejos valores de columnas de la tabla detalle, en el caso de actualización, lo viejo y lo nuevo, y, por último, en el caso de la inclusión, sólo los nuevos valores.

Figura 8

Estructura que prepara la tabla para ser replicada.

Fuente: Elaboración propia.

```
1   CREATE OR REPLACE FUNCTION f_detalle_transactions() RETURNS TRIGGER AS
    $f_detalle_transactions$
2   DECLARE
3   BEGIN
4   IF (TG_OP = 'DELETE') THEN
5   INSERT INTO detalle_transactions (
6   status, operation, id_tabla, OLD_entidad, OLD_fondo, OLD_clase,
7   OLD_recurso, criado)
8   VALUES (
9   'NEW', 'DELETE', OLD.id, OLD.nome, OLD.profissao, OLD.salario,
10  OLD.pagaAluguel, current_time);
11  ELSIF (TG_OP = 'UPDATE') THEN
12  INSERT INTO detalle_transactions (
13  status, operation, id_tabla, NEW_entidad, NEW_fondo, NEW_clase,
        NEW_recurso,
14  OLD_entidad, OLD_fondo, OLD_clase, OLD_recurso, criado)

15  VALUES (
16  'NEW', 'UPDATE', NEW.id, NEW.nome, NEW.profissao, NEW.salario, NEW.
    pagaAluguel,
17  OLD_entidad, OLD.profissao, OLD.salario, OLD.pagaAluguel, current_time);
18  ELSIF (TG_OP = 'INSERT') THEN
19  INSERT INTO detalle_transactions (
20  status, operation, id_tabla, NEW_entidad, NEW_fondo, NEW_clase,
21  NEW_recurso, criado)
22  VALUES (
23  'NEW', 'INSERT', NEW.id, NEW.entidad, NEW.fondo, NEW.clase,
24  NEW.recurso, current_time);
25  END IF;
26  RETURN NULL;
27  END;
28  $f_detalle_transactions$ LANGUAGE plpgsql;
29  CREATE TRIGGER t_detalle_transactions
30  AFTER INSERT OR UPDATE OR DELETE ON pacientes
31  FOR EACH ROW EXECUTE PROCEDURE f_detalle_transactions();
```

La estructura algorítmica para esta primera etapa viene dada de la siguiente forma:

Figura 9

Estructura algorítmica para la creación de un trigger síncrono recolector.

Fuente: Elaboración propia.

```
//------------------------------ CREACION TRIGGER SINCRONO RECOLECTOR-----------------------
//CREAR O REEMPLAZAR LA FUNCION PARA LA EVALUACION DE LAS TABLAS DE LA BASE DE DATOS
CREATE OR REPLACE FUNCTION trigger sincrono()
            //EN CASO DE EXISTENCIA BORRAR Y REEMPLAZAR
            //EN CASO DE NO EXISTIR CREAR UNA FUNCION CON EL SGTE. CODIGO
                DECLARE
                    BEGIN
                        //ACTUALIZAR TODO EL CONJUNTO DE DATOS ALMACENADOS EN LA TABLA
                        UPDATE nombre_de_la_tabla_en_evaluacion SET
                            campo1= NEW.campo1,
                            campo2= NEW.campo2,
                            campo3= NEW.campo3,
                            .
                            .
                            .
                            campoN= NEW.campoN,

                        //HACER TODO LO ANTERIOR SI Y SOLO SI idtabla=NEW.idtabla
                        WHERE id=NEW.id;
                        RETURN N ULL
                        //SINO RETORNAR NULL
FIN DE LA FUNCION
END;
LENGUAGE 'plpgsql'
VOLATILE COST 100;
ALTER FUNCTION trigger sincrono() OWNER TO postgres;
```

2.3.5.2 Mapeamiento

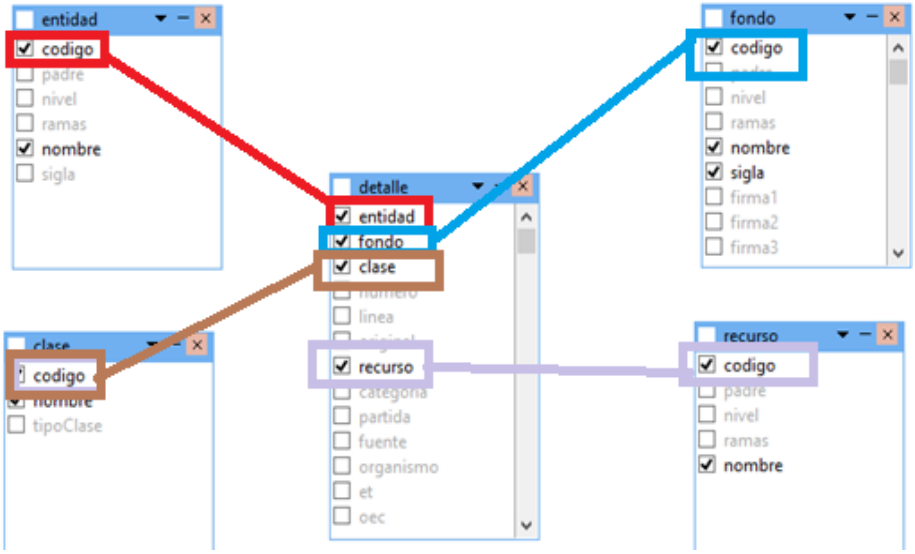
Antes de ejecutar el proceso de replicación es necesario crear un mapeo de las tablas de origen y de destino. Este paso lo podemos realizar con el mismo Sistema Gestor de Base de Datos, ya sea por línea de comandos o bajo la interfaz gráfica.

En este ejemplo utilizaremos el SGBD PostgreSQL,bajo su interfaz gráfica denominada PHPPGADMIN.

Figura 10

Relación de claves primarias.

Fuente: Elaboración propia.



La estructura algorítmica para esta segunda etapa viene dada de la siguiente forma:

Figura 11

Estructura algorítmica de un trigger asíncrono.

Fuente: Elaboración propia.

```
//------------------------------ CREACION TRIIGER ASINCRONO-----------------------------------------------------
//CREAR UNA SECUENCIA LLAMADA relacion_nuevos_datos_apuntando_a_los_viejos_datos
    CREATE SEQUENCE tabla_apuntandor_nuevo_a_los_viejos_datos
//CREAR LA TABLA EN BASE A LA ESTRUCTURA DE LA LINEA ANTERIOR LLAMADA apuntador
//DECLARAR UN CAMPO QUE TRABAJE COMO CLAVE PRIMARIA
    CREATE TABLE relacion_nuevos_datos_apuntando_a_los_viejos_datos(
        //declaramos el campo_llave_primaria, el estado de la tabla, la operacion a realizarse y la clave primaria de la tabla
        // tabla a replicar
        id integer NOT NULL DEFAULT NEXTVAL('relacion_nuevos_datos_apuntando_a_los_viejos_datos'),
        esatdo character varying(50) NOT NULL,
        operacion character varying(50) NOT NULL
        id_tabla integer NOT NULL,
        //EL CAMPO NUEVO VA TOMAR EL VALOR DE; CAMPO VIEJO
        NEW_campo1 integer,OLD_campo2 integer,// DONDE INTEGER ES EL TIPO DE DATO
        NEW_campo2 integer,OLD_campo2 integer,
        NEW_campo1 character varying(200),OLD_campo2 character varying(200),
        .
        .
        NEW_campoN tipo_de_dato,OLD_campoN tipo_de_dato,
        );
        CREATE UNIQUE INDEX tabla_apuntandor_nuevo_a_los_viejos_datos ON tabla_apuntandor_nuevo_a_los_viejos_datos(id);
        // CREAR FUNCION PARA SABER EL ESTADO DE LA VARIABLE OPERACION
        CREATE OR REPLACE FUNCTION funcion_tabla_apuntandor_nuevo_a_los_viejos_datos() RETURNS TRIGGERS AS
        $funcion_tabla_apuntandor_nuevo_a_los_viejos_datos$
        DECLARE
            BEGIN
                IF (TG_OP='DELETE') THEN
                    INSERT INTO tabla_apuntandor_nuevo_a_los_viejos_datos(
```

```
                    estado,operacion,id_tabla,OLD_campo1,OLD_campo2,....OLD_campoN )
                values (
                    'NEW','DELETE',OLD.id_tabla_VIEJO_CAMPO_A_BOORRAR,OLD.campo1_VIEJO_CAMPO_A_BOORRAR,
                    OLD.campo2_VIEJO_CAMPO_A_BOORRAR,.....OLD.campoN_VIEJO_CAMPO_A_BOORRAR,current time
                );
            ELSE IF (TG_OP='UPDATE') THEN
                INSERT INTO tabla_apuntandor_nuevo_a_los_viejos_datos(
                    estado,operacion,id_tabla,NEW.nuevo_campo1_ACTUALIZAR,NEW.nuevo_campo2_ACTUALIZAR,
                    ...NEW.nuevo_campoN_ACTUALIZAR,OLD.nuevo_campo1_ACTUALIZAR,OLD.nuevo_campo2_ACTUALIZAR,
                    ...OLD.nuevo_campoN_ACTUALIZAR )
                values(
                    'NEW','UPDATE',NEW_id_tabla,NEW.nuevo_campo1,NEW.nuevo_campo2,...NEW.nuevo_campoN,
                    OLD.id_tabla,OLD.nuevo_campo1,OLD.nuevo_campo2,...OLD.nuevo_campoN,current time);
            ELSE IF (TG_OP='INSERT') THEN
                INSERT INTO tabla_apuntandor_nuevo_a_los_viejos_datos(
                    estado,operacion,id_tabla,NEW.nuevo_campo1_INSERTAR,NEW.nuevo_campo2_INSERTAR,
                    ...NEW.nuevo_campoN_INSERTAR )
                values(
                    'NEW','UPDATE',NEW_id_tabla,NEW.nuevo_campo1,NEW.nuevo_campo2,...NEW.nuevo_campoN,
                     current time);
            END IF
        RETURN NULL
    END;
    $funcion_tabla_apuntandor_nuevo_a_los_viejos_datos$
    LENGUAGE plpgsql;
    CREATE TRIGGER t_apuntandor_nuevo_a_los_viejos_datos
    AFTER INSERT OR UPDATE OR DELETE ON nombre_de_la_tabla_a_replicar
        FOR EACH ROW EXECUTE PROCEDURE  funcion_tabla_apuntandor_nuevo_a_los_viejos_datos();
```

2.3.5.3 Ejecución

El proceso de replicación implementado en el prototipo leerá todas las filas de la tabla detalle_ transaccions con el valor NEW en la columna Estado.

Para cada una de estas líneas, el proceso descubre qué operación se debe replicar (operación de la columna) y construye la operación utilizando las columnas prefijadas con nuevos y viejos y clave id_table.

Una vez ejecutada la instrucción con la operación, simplemente ejecutarlo en la tabla de destino y actualizar el estado de la operación para EJECUTADO.

La estructura algorítmica para esta tercera y última etapa viene dada de la siguiente forma:

Figura 12

Estructura algorítmica de un trigger asíncrono en la etapa de ejecución

Fuente: Elaboración propia.

```
class JobProcess
  def self.run(job_id, alert = true)
    inicio = Time.now
    puts "Início"+ inicio.to_s
    job = Job.find_by_id(job_id)
    status = "SUCESS"
   tablemap = job.tablemap
    source_transaction = tablemap[:source] + "_transactions"
    target = tablemap[:target].to_s
    source_table = ActiveRecord::Base.connection.tables.sort.select {|table| not
```

```
        table[source_transaction].nil? }
    if source_table.blank?
        write_log("ERROR", "Source transaction table does not exist!", job)
        return
    end
    target_table = ActiveRecord::Base.connection.tables.sort.select {|table| not
        table[target].nil? }
    if target_table.blank?
        write_log("ERROR", "Target table does not exist!", job)
        return
    end
    newFields = ActiveRecord::Base.connection.columns(source_transaction).select {|f
ield|
        not field.name["new"].nil? }
    fieldsSource = []
    fieldsTarget = []
    for map in tablemap.columnmaps
      for sourcename in map.sourcecolumnmaps
        fieldsSource << sourcename.name.to_s
      end
      for targetname in map.targetcolumnmaps
        fieldsTarget << targetname.name.to_s
      end
    end
    while true
        transactions = ActiveRecord::Base.connection.select_all("select * from " +
                    source_transaction + " where status!='EXECUTED' order by id")
        if transactions.blank?
            job.status = "ESPERANDO"
            job.save!
            puts "PROCESO ESPERANDO"
        else
            job.status = "EJECUTANDO"
            job.save!
            puts "PROCESO EJECUTANDO"
            for transaction in transactions
              ActiveRecord::Base.connection.execute("UPDATE " + source_transaction +
                                  " set inicio_PROCESO = current_time where id=" +
                                      transaction["id"])
              comand = ""
              if transaction["operation"] == "INSERT"
                comand = "INSERT INTO " + target + "(" + (fieldsSource*",") +" , id)
                      values ("
                count = 1
                for field in fieldsSource
                  columns = newFields.select { |column| column.name[4.. column.name.siz
e]
                        == field }
                  if columns[0].type.to_s == "string" or columns[0].type.to_s == "datetime"
                    comand += "'" + transaction["new_" + field].to_s + "'"
                  else
                    comand += transaction["new_" + field].to_s
                  end
                  if count != fieldsSource.size
                    comand += ","
                  end
                  count += 1
```

```
60                  end
61                  comand += "," + transaction["id_table"].to_s + ")"
62                end
63                if transaction["operation"] == "DELETE"
64                  comand = "DELETE FROM "+ target + " WHERE id = "+
                         transaction["id_table"].to_s
65                end
66                if transaction["operation"] == "UPDATE"
67                  comand = "UPDATE "+ target + " SET ("+ (fieldsTarget*",") +") = ("
68                  count = 1
69                  for field in fieldsSource
70                    columns = newFields.select { |column| column.name[4..column.name.size]
                           == field }
71                    if columns[0].type.to_s == "string" or columns[0].type.to_s == "datetime"
72                      comand += "'"+ transaction["new_" + field].to_s + "'"
73                    else
74                      comand += transaction["new_" + field].to_s
75                    end
76                    if count != fieldsSource.size
77                      comand += ","
78                    end
79                    count += 1
80                  end
81                  comand += ") WHERE id = "+ transaction["id_table"].to_s
82                end
83                begin
84                  result = ActiveRecord::Base.connection.execute(comand)
85                rescue
86                  result = nil
87                  puts("==== Erro de lock ===")
88                end
89                if not result.nil?
90                  ActiveRecord::Base.connection.execute("UPDATE "+ source_transaction +
91                          " set status = 'EXECUTED', fim_PROCESO = current_time where id="
                              + transaction["id"])
92                end
93              end #--- LOOP DE TRANSACOES
94            end #--- IF BLANK
95          end #--- LOOP INFINITO
96        end
97        def self.write_log (status, message, job)
98          log = Log.new
99          log.type_log = status

100         log.description = message

101         log.job_id =job.id

102         log.save!

103         job.status = status

104         log.save!

105                                                                                    end
```

2.4 Experimentación

Con el fin de validar la propuesta presentada en esta tesis, se realizó un experimento para comparar la solución de replicación sincrónica frente a la solución asincrónica presentado en el capítulo anterior.

La comparación se hace mediante la comprobación en varios niveles de la replicación de base de datos, evaluando la concurrencia y el rendimiento de las dos soluciones.

2.4.1 Organización

De acuerdo a la nomenclatura presentada en el libro de Wohlin [WRH 00], el experimento que es adecuado para la comparación de métodos sincrónicos y asincrónicos para la refactorización de base de datos es el experimento de ingeniería.

Este tipo de experimento se define como el método para observar las soluciones existentes, sugiere las soluciones más adecuadas, desarrolla, medidas, análisis, y repite este proceso hasta que ya no es posible mejorar.

El autor de The Art of Computer Systems Analysis Performonce [JAI 91] Raj Jain, define varios pasos que se deben seguir en un experimento.

Entre ellos llevaremos a cabo las siguientes acciones:

2.4.1.1 Descripción del sistema y objetivos de la experiencia:

El primer paso propuesto por Raj Jain [JAI 91], es indicar los objetivos del experimento y definir lo que constituye el sistema, claramente delineando los límites del sistema.

Dado el mismo conjunto de hardware y software, la definición de sistema puede variar dependiendo los objetivos a alcanzar del estudio. En el caso de la replicación de base de datos, es necesario definir cuál será la aplicación cliente, que tablas se encuentran involucradas y que operaciones se ejecutaran en la base de datos (incluyendo, actualización, exclusión o consulta).

La elección de los límites del sistema, afectará a la definición de métricas de desempeño, así como cargas de trabajo necesarias para comparar los sistemas.

2.4.1.2 Selección de métricas:

El siguiente paso es la selección de criterios para comparar el rendimiento entre los sistemas. Estos criterios se denominan métricas y están relacionados con la velocidad, exactitud y disponibilidad de servicios.

En el caso de un experimento en una red de datos, el rendimiento se mide por la velocidad (caudal y retardo), la precisión (tasa de error), y la disponibilidad de los paquetes enviados.

El rendimiento de una base de datos se mide por la velocidad (el tiempo necesario para llevar a cabo) varias operaciones (consulta, insertar, actualizar y eliminar).

2.4.1.3 Criterios de selección:

Debe hacer una lista de todos los parámetros que afectan al rendimiento. La lista se puede dividir en parámetros del sistema y los parámetros de la carga de trabajo.

Los parámetros del sistema incluyen definir el hardware y el software utilizado en el entorno de la prueba.

Los parámetros de carga de trabajo son características de los clientes solicitan a la base de datos, como el número de clientes, incluyendo las operaciones realizadas, el número de operaciones que se realizan dentro de una transacción, etc.

2.4.1.4 Desarrollo del experimento

Después de todos los pasos anteriores, usted tendrá que decidir sobre una serie de experimentos que proporcionan la mayoría de la información del entorno analizado con un mínimo esfuerzo.

¿Cuántas repeticiones del experimento son necesarias? ¿Qué parámetros deben modificarse o mantenerse en cada una de estas repeticiones? ¿Cómo se hizo para recopilar las métricas?

Todas estas preguntas deben ser contestadas para iniciar el experimento.

2.4.1.5 Análisis e interpretación de los datos

Es importante tener en cuenta que los resultados de las mediciones y simulaciones son cantidades aleatorias que el resultado sea diferente cada vez que se repite el experimento.

Por lo tanto, cuando se comparan dos alternativas, es necesario tratar la variabilidad de los resultados.

Comparar los datos recogidos sin utilizar ninguna técnica estadística, puede llevar a conclusiones erróneas. Interpretación de los resultados es el paso más importante del experimento, es en este punto que producen los resultados.

2.4.1.6 Presentación de los resultados

La etapa final del experimento es el de presentar los resultados. Es importante que los resultados se presenten de forma clara.

2.4.2 Descripción del sistema y de los objetivos del experimento

El ambiente de “trabajo” ha sido elegido para llevar a cabo el experimento, de forma sencilla y representativa, de tal forma que nos permita la comparación entre las soluciones.

La base de datos, sobre la cual trabajaremos, se encuentra incluida en la sección de anexos A-1 la misma que trata de los estados financieros de una Institución Pública del Estado Plurinacional de Bolivia, por motivos de seguridad de la información, nos reservamos dar el nombre de la Institución; la misma que entregó la base de datos, al investigador (autor de esta tesis) , bajo términos y normas de uso académico.

Además se vio por conveniente el uso de una sola tabla de la base de datos, ya que la misma, es la tabla principal, que anexa cada una de las demás tablas a partir de relaciones que existen de claves foráneas.

Esta tabla contiene un total de 18759 registros y lleva como nombre detalle, la misma que está incluida en la sección anexos A-1.

El objetivo del experimento es para validar la hipótesis siguiente:

2.4.2.1 Hipótesis principal

La aplicación de algoritmos evolutivos asíncronos, ayuda a mejorar la lógica de transferencia de datos en los procesos de replicación de bases de datos transaccionales grandes .

A partir de esta hipótesis principal, podemos deducir las siguientes variables, tales como:

- Variable 1: Tiempo.- El tiempo pendiente necesario para procesar un método de operación asincrónica utilizando algoritmos evolutivos, es pequeño y está en el orden de decenas de milisegundos.

- Variable 2: Bloqueos.- El método sincrónico de replicación de bases de datos sin la utilización de algoritmos evolutivos, genera muchos bloqueos, la cantidad es mayor que el método asincrónico, utilizando algoritmos evolutivos.

- Variable 3: Rendimiento.- La actualización usando un método de replicación asíncrona, tiene un rendimiento, medido por el número de operaciones que se actualiza por segundo, mayor que cuando se utiliza el método sincrónico.

Las variables 1 y 2 tienen como foco principal, la realización de la comparación entre los métodos sincrónicos y asincrónicos en términos de rendimiento.

El primero mirando la cantidad de bloqueos y el segundo mirando para el número de operaciones por segundo.

La variable 3 se refiere al período de que la tabla en estudio es inconsistente, porque sólo después de la ejecución del proceso de replicación es que no habrá más inconsistencia.

El tiempo promedio de procesamiento de una operación pendiente proporcionará una estimación del período de falta de coherencia en la tabla.

2.4.3 Selección de métricas

Las métricas sirven en un experimento para verificar las hipótesis en el entorno experimental.

La recolección y el análisis de los valores de los parámetros, permiten decidir si las hipótesis del experimento son válidas.

Los siguientes indicadores fueron seleccionados para ser observados durante nuestro experimento:

- Medida 1 - Número de bloques: la base de datos, para el control de la concurrencia por el acceso a la misma fila de la tabla, utiliza bloques que interrumpen la ejecución de un proceso para que otro termine o se ejecute mientras tanto.

Considerando que los procesos son aleatorios, para la elección de que las filas de una tabla se actualizarán y si nos fijamos en el número de filas en la tabla, obviamente mayor será el número de procesos, y mayor es la probabilidad de que se produzcan bloqueos.

En consecuencia, en esta situación, el rendimiento promedio de todos los procesos es de disminuir.

- Métricas 2 - Número de transacciones por segundo: el número de operaciones realizadas por el gestor de base de datos en un intervalo de tiempo dividido por el tiempo en segundos es el valor de este indicador.

Esta métrica se utiliza para medir el rendimiento medio de los procedimientos que realizan estas operaciones en la base de datos.

- Métricas 3 - La inconsistencia temporal: en un ambiente con la replicación asincrónica, la inconsistencia en el tiempo se puede estimar el número de transacciones pendientes versus el tiempo medio necesario para procesar una transacción en la tabla de destino.

La métrica 1 se dirige a validar la variable 2.

La 2da métrica es útil para validar la 3ra variable..

Finalmente, la métrica 3 se preparó para que podamos verificar la variable 1.

A través de esta organización, tenemos lo suficiente para hacer las mediciones experimentales.

2.4.4 Selección de los parámetros

Tenemos dos tipos de parámetros para crear un entorno de laboratorio que es lo más cercano a lo real.

El primero son los parámetros que definen el hardware y software del sistema.

El segundo son los parámetros de carga del sistema que controlan la captación y acceso entorno competitivo.

Las secciones siguientes describen estos dos tipos de parámetros.

2.4.5 Preparación de los experimentos

Se han desarrollado tres escenarios para poner a prueba las hipótesis propuestas. Todos los escenarios tienen el mismo proceso de generación de las operaciones simultáneas y trabajan bajo la misma tabla. Lo que diferencia a los escenarios es la existencia de triggers y su tipo: asíncrono o síncrono.

2.4.5.1 Escenarios

Se crearon dos tablas de prueba llamadas detalle_0 y detalle_1, con la misma estructura pero con diferentes nombres. Cuando hay replicación de datos, la tabla detalle_0 es la fuente de datos y la tabla detalle_1 es el destino.

A continuación se muestra una descripción de cada escenario:

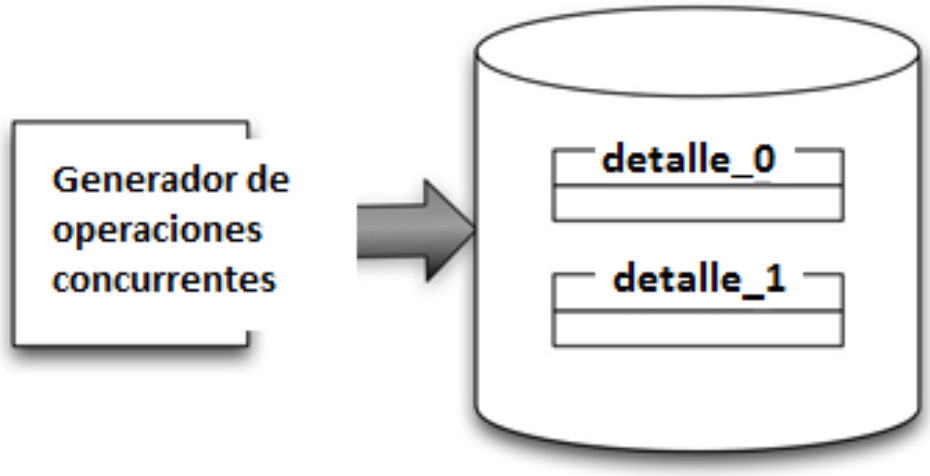
- Escenario 1 - No triggers: las dos tablas de prueba tienen la replicación de datos, es decir, no hay ningún trigger asociado con dos tablas.

Este escenario tiene como objetivo determinar cuál es el más alto rendimiento del entorno de simulación.

Figura 13

Escenario sin triggers

Fuente: Elaboración propia.

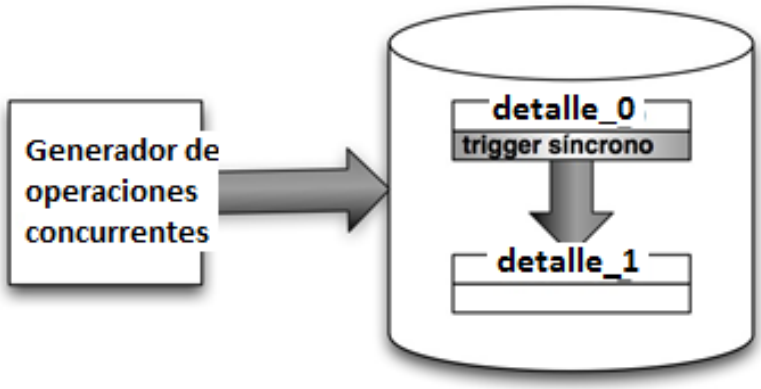


- Escenario 2 - Trigger sincrónico: El propósito de este escenario es el de evaluar el rendimiento de la solución actual para la replicación de datos en refactorizaciones base de datos.

Figura 14

Escenario con trigger síncrono.

Fuente: Elaboración propia.



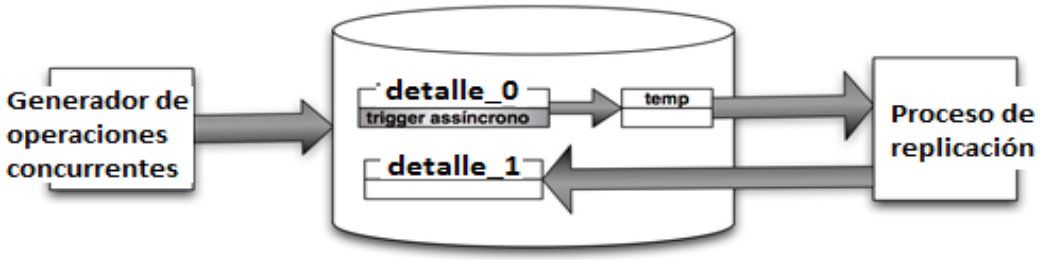
- Escenario 3 – Trigger asíncrono: este escenario tiene como objetivo el evaluar las bases de datos de replicación asincrónica bajo las hipótesis propuestas.

La figura 15 muestra las tablas involucradas y triggers asincrónicos, grabando los datos de las operaciones de registro en una tabla temporal (temp) y el proceso de replicación que se ejecutan simultáneamente con el proceso de generador de operaciones concurrentes.

Figura 15

Escenarios con triggers asíncronos.

Fuente: Elaboración propia.



CAPÍTULO 3

MARCO PRÁCTICO

El presente capítulo, describe los resultados de la implementación de la estructura algorítmica propuesta, a través de la experimentación realizada en una base de datos grande, la cual almacena alrededor de 18000 registros; posteriormente se demuestra gráfica y estadísticamente los resultados y por último se hace la interpretación de los mismos..

3.1 Prototipo

Para validar la hipótesis propuesta en la presente tesis denominada, Replicación asincrónica de base de datos utilizando algoritmos evolutivos, se realizó implementación de un prototipo funcional, el cual utiliza como sistema gestor de base de datos PostgresSQL y como lenguaje de manipulación de datos el estándar SQL.

A la par, se utiliza un aplicativo, el cual permite a los usuarios realizar procesos de replicación entorno a un determinado algoritmo de replicación.

En pocas palabras, permite al administrador de la base de datos, realizar la inserción de un algoritmo al aplicativo, para que , bajo los parámetros del mismo, se evalué los procesos de replicación de base de datos , obviamente asignándole un origen y un destino.

Cabe destacar, que no es del todo necesario, utilizar un aplicativo para realizar estos procesos, ya que PostgresSQL, cuenta con un módulo similar al que utilizaremos para procesos de replicación, pero obviamente, tiene una interfaz mucho más técnica, para realizar algunas tareas es necesario utilizar la consola del sistema operativo, se necesita mucho más conocimiento del sistema gestor de base de datos y conocer muchas más el lenguaje de manipulación.

A diferencia del aplicativo, que básicamente tiene las mismas tareas asignadas, pero esta vez automatizadas, y más entendibles visualmente.

Algunas desventajas entorno al uso del aplicativo, es la incompatibilidad actual con múltiples sistemas operativos.

3.1.2 Pasos para realizar el experimento:

3.1.2.1 Preparación del ambiente

Esta etapa consta, en describir todos aquellos requerimientos tanto físicos, como lógicos, necesarios para la realización del experimento.

3.1.2.1.1 Requerimientos físicos

Para la realización del prototipo funcional de la siguiente investigación, se hizo uso de tres servidores reales en línea, pertenecientes a una oficina gubernamental del Estado Plurinacional de Bolivia, mismos que son alojados en repositorios internacionales AWS AMAZON.

3 Servidores :

Procesador : Intel Xeon Family t2.micro

Memoria ram : 1GB

Velocidad : 2.5 GHz

Disco duro : 8GB

3.1.2.1.2 Requerimientos lógicos

Sistema operativo : Ubuntu Server 14.04 LTS (HVM), SSD Volume Type – ami-29ebb519

Puertos de comunicación : SSH,HTTP,HTTPS,5432

Sistema gestor de base de datos : PostgreSQL 9.3

Herramienta de replicación : Bucardo

Manejador de base de datos : PGADMIN3

Lenguajes de programación : PERL ,PHP y PLSQL

Lenguajes de consultas : SQL

Sistema gestor de base de datos :. Versión PostgreSQL Manager 8.4 se utilizó con una instalación predeterminada. El único parámetro cambiado es el número máximo de conexiones. El valor configurado era 200, debido a la cantidad de memoria de la máquina virtual. Cuando se trata de configurar un valor mayor de conexiones, por ejemplo 250, se produjo un error en PostgreSQL que requiere un aumento en el número de memoria compartida del sistema operativo. No se realizó esta reconfiguración del sistema operativo porque el objetivo es tener una instalación estándar y, por lo tanto, alterar el mínimo posible las instalaciones de software originales.

Por lo tanto, asumimos que el límite es 200 para el número máximo de conexiones.

En el anexo A se presenta la configuración completa de PostgreSQL.

3.1.2.1.3 ¿Por qué PostgreSQL?

- Código fuente libre y de alta calidad
- Licencia BSD - En pocas palabras, puedes hacer prácticamente lo que quieras con el producto, sin restricciones.
- Soporte profesional tanto de la comunidad como de empresas especializadas.
- Requerimientos de administración y mantenimiento bajos con respecto el resto de bases de datos comerciales
- Fiabilidad y estabilidad legendarias
- Rendimiento excelente
- Diseñada para entornos con altos volúmenes de tráfico/transacciones
- Extensible
- Multiplataforma
- Herramientas gráficas y de línea de comandos para diseñar nuestras bases de datos y administrarlas.

Actualmente las empresas gubernamentales del Estado Plurinacional de Bolivia, están optando por usar software de código abierto, por lo tanto se busca herramientas tecnológicas que soporten altos volúmenes de tráfico de información, tal como lo describe el punto 7 citado anteriormente.

3.1.2.2 Recolección de la información

Para esta primera etapa de la estructura algorítmica, vamos a realizar una copia de los nuevos y viejos datos de las tablas a replicar; para ello ejecutaremos el siguiente script en nuestro SGBD , o bien en un manejador de base de datos.

Figura 16

Copias de respaldo (backups) de las tablas

Fuente: Elaboración propia.

```
select * into arbolestrato_nuevo from arbolestrato;
select * into cambio_nuevo from cambio;
select * into cambiomoneda_nuevo from cambiomoneda;
select * into cambious_nuevo from cambious;
select * into detalle_nuevo from detalle;
select * into forma_nuevo from forma;
```

Posteriormente ejecutamos el denominado ‘trigger_recolector ’ para cada una de las tablas a replicarse.

En este ejemplo, solo ejecutaremos para tres tablas .

Figura 17

Creación del trigger recolector

Fuente: Elaboración propia.

```
CREATE OR REPLACE FUNCTION recolector_arbolestrato() RETURNS TRIGGER AS $trigger$
DECLARE BEGIN
        INSERT INTO arbolestrato_nuevo("codigo", "padre", "nivel", "ramas", "nombre") select * from arbolestrato;
        RETURN NULL;
END;
$trigger$LANGUAGE plpgsql;

CREATE TRIGGER recolector_arbolestrato AFTER INSERT OR UPDATE
ON arbolestrato FOR EACH ROW
EXECUTE PROCEDURE recolector_arbolestrato();

CREATE OR REPLACE FUNCTION recolector_cambio() RETURNS TRIGGER AS $trigger$
DECLARE BEGIN
        INSERT INTO cambio_nuevo ("Entidad", "Fondo", "Clase", "Numero", "Moneda", "Cambio", "pkcambio") select * from cambio;
        RETURN NULL;
END;
$trigger$LANGUAGE plpgsql;

CREATE TRIGGER recolector_cambio AFTER INSERT OR UPDATE
ON cambio FOR EACH ROW
EXECUTE PROCEDURE recolector_cambio();

CREATE OR REPLACE FUNCTION recolector_cambiomoneda() RETURNS TRIGGER AS $trigger$
DECLARE BEGIN
        INSERT INTO cambiomoneda_nuevo ("Moneda", "fecha", "Cambio", "pkcambiomoneda") select * from cambiomoneda;
        RETURN NULL;
END;
$trigger$LANGUAGE plpgsql;

CREATE TRIGGER recolector_cambiomoneda AFTER INSERT OR UPDATE
ON cambiomoneda FOR EACH ROW
EXECUTE PROCEDURE recolector_cambiomoneda();
```

3.2.3 Mapeamiento

En esta segunda etapa de la estructura algorítmica, el objetivo es verificar si todas las tablas a replicarse contienen una clave primaria (obligatoriamente) y una foránea (opcional).

Figura 18

Verificación de las claves primarias

Fuente: Elaboración propia.

```
ALTER TABLE arbolestrato ADD COLUMN pk_arbolestrato serial
ALTER TABLE cambio ADD COLUMN pk_cambio serial
ALTER TABLE cambiomoneda ADD COLUMN pk_cambiomoneda serial
ALTER TABLE cambious ADD COLUMN pk_cambious serial
ALTER TABLE detalle ADD COLUMN pk_detalle serial
ALTER TABLE forma ADD COLUMN pk_forma serial
```

En caso de que no exista una clave primaria, para una determinada tabla a replicarse, se tendrá que ejecutar el siguiente script de asignación de clave primaria, que se muestra en la figura 19.

Figura 19

Asignación de clave primaria a una tabla.

Fuente: Elaboración propia.

```
-- Primary Key structure for table ArbolEstrato
-- ----------------------------
ALTER TABLE "public"."ArbolEstrato" ADD PRIMARY KEY ("codigo");

-- ----------------------------
-- Primary Key structure for table cambio
-- ----------------------------
ALTER TABLE "public"."cambio" ADD PRIMARY KEY ("pkcambio");

-- ----------------------------
-- Primary Key structure for table CambioMoneda
-- ----------------------------
ALTER TABLE "public"."CambioMoneda" ADD PRIMARY KEY ("pkcambiomoneda");

-- ----------------------------
-- Primary Key structure for table CambioUs
-- ----------------------------
ALTER TABLE "public"."CambioUs" ADD PRIMARY KEY ("pkcambious");

-- ----------------------------
-- Primary Key structure for table detalle
-- ----------------------------
ALTER TABLE "public"."detalle" ADD PRIMARY KEY ("pkdetalle");

-- ----------------------------
```

3.1.2.4 Ejecución

Para la ejecución de la tercera etapa de la estructura algorítmica, se supone que ya han sido ejecutados los pasos 1, 2, y 3 (preparación del ambiente, recolección de información y mapeamiento) del presente capitulo.

Para la siguiente experimentación, manejaremos una estructura de replicación maestro maestro; por lo que todos los nodos incluidos en el proceso se replicarán del nodo origen al nodo destino y viceversa, con el fin de mantener ambos nodos actualizados.

Para ello procedemos a la carga de la estructura algorítmica, vista en la figura 12 del capítulo anterior, al aplicativo de procesos de replicación basados en PostgreSQL denominado bucardo y continuamos con la configuración de nuestros nodos que serán participes de la replicación.

Para ello ejecutamos el siguiente script, en una línea de comandos (terminal) Linux.

Figura 20

Etapa de ejecución

Fuente: Elaboración propia.

```
Archivo  Edición  Formato  Ver  Ayuda
bucardo install --piddir=/tmp
------------------AÑADIENDO LAS BASES DE DATOS A REPLICARSE----------------
bucardo add database server1 dbname=bdtesis
bucardo add database server2 dbname=bdtesis host=54.149.62.102 user=bucardo password=mija16291 port=5432
------------------AÑADIENDO LAS TABLAS A REPLICARSE----------------
bucardo add table cambio db=server1
bucardo add table cambio db=server2
------------------AÑADIENDO GRUPOS DE REPLICACION-----------------------------------
bucardo add herd server-a_server-b cambio db=server1
bucardo add herd server-b_server-a cambio db=server2
-----------------DEFINIENDO ORIGEN Y DESTINO-------------------------------------
bucardo add sync bdtesis_sync_a relgroup=server-a_server-b dbs=server1,server2
bucardo add sync bdtesis_sync_b relgroup=server-b_server-a dbs=server2,server1
-------------------EJECUTANDO-----------------------------------------------
bucardo start
```

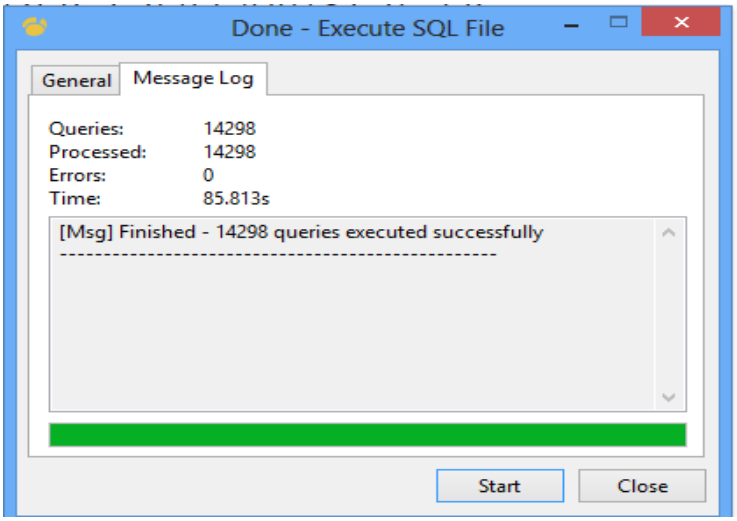
Una vez ejecutado el script, las bases de datos y por ende las tablas participes de la replicación, se encuentran conectadas entre si, por lo tanto podemos proseguir con la verificación.

Para verificar que se está ejecutando nos vamos a nuestro SGBD y realizamos cambios a las tablas involucradas en la replicación, en este caso la tabla cambio.

Figura 21

Verificación en nuestro SGBD.

Fuente: Elaboración propia.



Hacemos una consulta, a la tabla cambio en nuestro nodo origen.

Figura 22

Consulta de verificación.

Fuente: Elaboración propia.

```
1  select count(pkcambio) from cambio
```

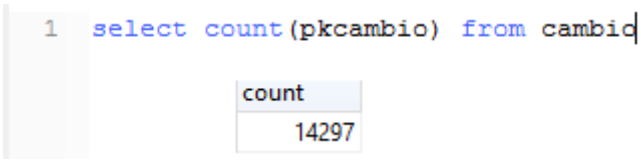
count
14297

Verificamos, si la replicación se está ejecutando.

Figura 23

Resultado de la consulta de verificación.

Fuente: Elaboración propia.



La replicación se ejecutó con éxito.

CAPÍTULO 4

PRUEBA DE HIPÓTESIS

El presente capítulo, describe los resultados de la implementación de la estructura algorítmica propuesta, a través de la experimentación realizada en una base de datos grande, la cual almacena alrededor de 18000 registros; posteriormente se demuestra gráfica y estadísticamente los resultados y por último se hace la interpretación de los mismos..

4.1 Análisis e interpretación de los datos

En esta sección se presentan y analizan los resultados obtenidos en el experimento.

Para ello utilizaremos el método estadístico “coeficiente kappa’, para determinar si nuestra hipótesis principal es verdadera o falsa.

Por lo tanto se determinaran cuatro variables fundamentales, según el coeficiente de kappa, para la evaluación del desempeño del algoritmo, analizados bajo los siguientes parámetros:

Figura 24

Parámetros de interpretación KAPA

Fuente: [VIL 2008]

Índice Kappa	Interpretación
0.00 – 0.20	Ínfima concordancia
0.20 – 0.40	Escasa concordancia
0.40 – 0.60	Moderada concordancia
0.60 – 0.80	Buena concordancia
0.80 – 1.00	Muy Buena concordancia

Entonces, analizaremos cada una de las variables, anexas a la hipótesis principal, y experimentaremos su efectividad para cada uno de los escenarios de trabajo.

En primer lugar, realizaremos el cálculo manual para un mejor entendimiento, posterior a ello, realizaremos el cálculo automáticamente utilizando un script desarrollado en php, elaborado por mi persona, que básicamente ayuda en automatizar los procesos de cálculo del coeficiente Kappa.

Entonces, para realizar el cálculo manualmente previamente debemos entender que:

Figura 25

Fórmula coeficiente KAPA.

Fuente: [VIL 2008]

$$K = \frac{p_0 - p_e}{1 - p_e}$$

siendo: $$p_0 = \frac{núm.acuerdos}{num.acuerdos + num.desacuerdos}$$

$$p_e = \sum_{i=1}^{n} (p_{i1} \times p_{i2})$$

Donde:

n= número de categorías

i=número de la categoría

p_{i1} = proporción de ocurrencia de la categoría i para el observador 1

p_{i2} = proporción de ocurrencia de la categoría i para el observador 2

4.1.1 Variable 1: Tiempo

Analizaremos el coeficiente kappa, entorno a la cantidad de procesos ingresados contra la cantidad de tiempo que tarda en procesar, para cada uno de los escenarios, bajo los siguientes parámetros de evaluación:

Tabla 1

Parámetros de evaluación para el coeficiente kappa, entrono a las variables presentadas en la hipótesis..

PARÁMETROS DE EVALUACIÓN PARA EL MODELO KAPPA APLICADO A LAS VARIABLES DE LA HIPOTESIS PRESENTADA EN ESTA INVESTIGACIÓN		
CASO	**TIEMPOS**	**CALIFICACIN**
A	0-20	5
B	20-40	4
C	40-60	3
D	60-80	2
E	80-100	1

Donde 5 es un buen tiempo y 1 es el peor tiempo en procesos de replicación expresado en segundos.

4.1.1.1 Análisis del coeficiente Kappa en un escenario sin triggers

Para este primer caso de análisis, se realizó 12 pruebas, cada una con un número determinado de procesos y un tiempo expresado en segundos, en un escenario sin triggers, misma que devolvió los siguientes resultados:

Tabla 2

Cantidad de procesos y medidas en tiempo expresado en segundos, en un escenario sin triggers.

Fuente: Elaboración propia.

ESCENARIO : SIN TRIGGERS		
PRUEBA 1		PRUEBA 2
PROCESOS	TIEMPO 1	TIEMPO 2
1000	30	35
1500	44	43
1800	50	53.7
2000	52	55
5000	53	61.23
10000	58	65

Posteriormente procedemos a calificar los tiempos de cada uno de los casos de prueba, entorno a los parámetros de calificacion mencionados en la tabla 1.

Tabla 3

Cantidad de procesos y medidas en tiempo expresado en segundos, en un escenario sin triggers.

Fuente: Elaboración propia.

ESCENARIO : SIN TRIGGERS				
PRUEBA 1			PRUEBA 2	
PROCESOS	TIEMPO	CALIFICACION	TIEMPO	CALIFICACION
1000	30	4	35	4
1500	44	3	43	3
1800	50	3	53.7	3
2000	52	3	53.7	3
5000	53	3	61.23	2
10000	58	3	65	2

Una vez calificados,procedemos a la estructuracion de nuestra tabla de concurrencia de casos, o sea cuantas veces se repitieron los mismos tiempos.

Tabla 4

Tabla de casos de concurrencia, entorno a tiempos, para los casos de prueba 1 y 2.

Fuente: Elaboración propia.

ESCENARIO :	SIN TRIGGERS					
	CALIFICACION					
TIEMPO 1	4	3	3	3	3	3
TIEMPO 2	4	3	3	3	2	2

Cálculo de Po :

P_0= (4)/(4+2)=0.67

Tabla 5

Reordenación de la tabla de concurrencia entorno a las demás calificaciones.

Fuente: Elaboración propia.

	1	2	3	4	5	Σ
1	0	0	0	0	0	**0**
2	0	0	2	0	0	**0.33**
3	0	0	3	0	0	**0.5**
4	0	0	0	1	0	**0.17**
5	0	0	0	0	0	**0**
Σ	**0**	**0**	**0.83**	**0.17**	**0**	

Cálculo de Pe:

Pe=(0.*0)+(0*0.33)+(0.5*0.83)+(0.17*0.67)+(0*0)

Pe=0.44

Cálculo del coeficiente kappa:

K=0.67-0.44/1-0.44

K=0.4

Entonces el valor del coeficiente kappa, en un escenario sin trigger es 0.4 o sea 40% de efectividad.

Continuamos con el análisis para los siguientes escenarios, con los mismos casos de prueba.

4.1.1.2 Análisis del coeficiente Kappa en un escenario con triggers síncronos

Tabla 6

Cantidad de procesos y medidas en tiempo expresado en segundos, en un escenario con triggers síncronos.

Fuente: Elaboración propia.

PRUEBA 1		PRUEBA 2
PROCESOS	TIEMPO 1	TIEMPO 2
1000	60.21	55.4
1500	58.2	59.6
1800	60.23	62.3
5000	48.7	45.25
2000	44.2	44.6
10000	41.6	39.1

Tabla 7

Cantidad de procesos y medidas en tiempo expresado en segundos, en un escenario con triggers síncronos.

Fuente: Elaboración propia.

ESCENARIO : TRIGGERS SINCRONOS				
PRUEBA 1			PRUEBA 2	
PROCESOS	TIEMPO	CALIFICACION	TIEMPO	CALIFICACION
1000	60.21	2	55.4	3
1500	58.2	3	59.6	3
1800	60.23	2	62.3	2
5000	48.7	3	45.25	3
2000	44.2	3	44.6	3
10000	41.6	3	39.1	4

Tabla 8

Tabla de casos de concurrencia, entorno a tiempos, para los casos de prueba 1 y 2.

Fuente: Elaboración propia.

ESCENARIO :	TRIGGERS SINCRONOS					
	CALIFICACION					
TIEMPO 1	2	3	2	3	3	3
TIEMPO 2	3	3	2	3	3	4

Tabla 9

Reordenación de la tabla de concurrencia entorno a las demás calificaciones.

Fuente: Elaboración propia.

	1	2	3	4	5	Σ
1	0	0	0	0	0	**0**
2	0	1	0	0	0	**0.17**
3	0	1	3	0	0	**0.67**
4	0	0	1	0	0	**0.17**
5	0	0	0	0	0	**0**
Σ	**0**	**0.33**	**0.67**	**0**	**0**	

Cálculo de Po

Po=0.67

Cálculo de Pe

Pe=0.5

Cálculo coeficiente Kappa

k=0.33

4.1.1.3 Análisis del coeficiente Kappa en un escenario utilizando triggers asíncronos

Tabla 10

Cantidad de procesos y medidas en tiempo expresado en segundos, en un escenario con triggers asíncronos.

Fuente: Elaboración propia.

ESCENARIO : TRIGGERS ASINCRONOS		
PRUEBA 1		PRUEBA 2
PROCESOS	TIEMPO 1	TIEMPO 2
1000	46.4	45
1500	46	46.23
1800	44	43.25
2000	40.4	41.2
5000	36	33
10000	31.5	32

Tabla 11

Cantidad de procesos y medidas en tiempo expresado en segundos, en un escenario con triggers asíncronos.

Fuente: Elaboración propia.

ESCENARIO : TRIGGERS ASINCRONOS				
PRUEBA 1			PRUEBA 2	
PROCESOS	TIEMPO	CALIFICACION	TIEMPO	CALIFICACION
1000	46.4	3	45	3
1500	46	3	46.23	3
1800	44	3	43.25	3
2000	40.4	3	41.2	3
5000	36	4	33	4
10000	31.5	4	32	4

Tabla 12

Tabla de casos de concurrencia, entorno a tiempos, para los casos de prueba 1 y 2.

Fuente: Elaboración propia.

ESCENARIO :	TRIGGERS ASINCRONOS					
	CALIFICACION					
TIEMPO 1	3	3	3	3	4	4
TIEMPO 2	3	3	3	3	4	4

Tabla 13

Reordenación de la tabla de concurrencia entorno a las demás calificaciones.

Fuente: Elaboración propia.

	1	2	3	4	5	Σ
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	3	0	0	0.5
4	0	0	0	2	0	0.33
5	0	0	0	0	0	0
Σ	0	0	0.5	0.33	0	

Cálculo de Po

Po=1

Cálculo de Pe

Pe=0.36

Cálculo coeficiente Kappa

k=1

Por lo tanto, analizando cada una de los resultados para el coeficiente kappa, asignados a un determinado escenario, podemos ver que el escenario asíncrono es el mejor entre los tres, ya que no presenta incoherencia en Po, debido a que los tiempos disminuyeron , y esto hace que todos los casos pertenezcan a una calificación tipo 3 o 4.

El coeficiente kappa para un escenario asincrono es 1, osea 100%, por lo tanto podemos decir que la variable 1 tiempo es verdadera.

4.1.2 Variable 2: Bloqueos

Ahora analizaremos el coeficiente kappa, entorno a la cantidad de bloqueos, para cada uno de los escenarios, bajo los parámetros de evaluación presentados en la tabla 1.

4.1.2.1 Análisis del coeficiente Kappa en un escenario sin triggers

Tabla 14

Cantidad de procesos y bloqueos expresados en porcentaje, en un escenario sin triggers.

Fuente: Elaboración propia.

ESCENARIO : SIN TRIGGERS		
PRUEBA 1		PRUEBA 2
PROCESOS	BLOQUEOS 1 %	BLOQUEOS 2 %
1000	20	19.21
1500	44	43.25
1800	17.6	18.21
2000	26.3	23.21
5000	15.21	14.56
10000	12.54	15.15

Tabla 15

Cantidad de procesos y bloqueos expresados en porcentaje, en un escenario sin triggers.

Fuente: Elaboración propia.

ESCENARIO : SIN TRIGGERS				
PRUEBA 1			PRUEBA 2	
PROCESOS	BLOQUEOS 1 %	CALIFICACION	BLOQUEOS 2 %	CALIFICACION
1000	20.54	4	16.21	5
1500	44	3	43.25	3
1800	17.6	5	18.21	5
2000	26.3	4	23.21	4
5000	15.21	5	14.56	5
10000	12.54	5	15.15	5

Tabla 16

Tabla de casos de concurrencia, entorno a tiempos, para los casos de prueba 1 y 2.

Fuente: Elaboración propia.

ESCENARIO :	SIN TRIGGERS					
	CALIFICACION					
BLOQUEOS 1 %	4	3	5	4	5	5
BLOQUEOS 2 %	5	3	5	4	5	5

Tabla 17

Reordenación de la tabla de concurrencia entorno a las demás calificaciones.

Fuente: Elaboración propia.

	1	2	3	4	5	Σ
1	0	0	0	0	0	**0**
2	0	0	0	0	0	**0**
3	0	0	1	0	0	**0.17**
4	0	0	0	1	0	**0.17**
5	0	0	0	1	3	**0.67**
Σ	**0**	**0**	**0.17**	**0.33**	**0**	

Cálculo de Po

Po=0.67

Cálculo de Pe

Pe=0.08

Cálculo coeficiente Kappa

k=0.64

4.1.2.2 Análisis del coeficiente Kappa en un escenario con triggers síncronos

Tabla 18

Cantidad de procesos y bloqueos expresados en porcentaje, en un escenario con triggers síncronos.

Fuente: Elaboración propia.

ESCENARIO : SINCRONOS		
PRUEBA 1		PRUEBA 2
PROCESOS	BLOQUEOS 1 %	BLOQUEOS 2 %
1000	11.65	11.02
1500	20.51	20.36
1800	25.41	24.12
2000	15.85	18.25
5000	20.62	21.32
10000	22.36	22.54

Tabla 19

Cantidad de procesos y bloqueos expresados en porcentaje, en un escenario con triggers síncronos.

Fuente: Elaboración propia.

ESCENARIO : SINCRONOS				
PRUEBA 1			PRUEBA 2	
PROCESOS	BLOQUEOS 1 %	CALIFICACION	BLOQUEOS 2 %	CALIFICACION
1000	11.65	5	11.02	5
1500	20.51	4	19.85	5
1800	23.41	4	24.12	4
2000	15.85	5	18.25	5
5000	20.62	4	21.32	4
10000	19.97	5	22.14	4

Tabla 20

Tabla de casos de concurrencia, entorno a tiempos, para los casos de prueba 1 y 2.

Fuente: Elaboración propia.

ESCENARIO :	SINCRONOS					
	CALIFICACION					
BLOQUEOS 1 %	5	4	4	5	4	5
BLOQUEOS 2 %	5	5	4	5	4	4

Tabla 21

Reordenación de la tabla de concurrencia entorno a las demás calificaciones.

Fuente: Elaboración propia.

	1	2	3	4	5	Σ
1	0	0	0	0	0	**0**
2	0	0	0	0	0	**0**
3	0	0	0	0	0	**0**
4	0	0	0	2	1	**0.5**
5	0	0	0	1	2	**0.5**
Σ	**0**	**0**	**0**	**0.5**	**0**	

Cálculo de Po

Po=0.67

Cálculo de Pe

Pe=0.25

Cálculo coeficiente Kappa

k=0.56

4.1.2.3 Análisis del coeficiente Kappa en un escenario utilizando triggers asíncronos

Tabla 22

Cantidad de procesos y bloqueos expresados en porcentaje, en un escenario con triggers asíncronos.

Fuente: Elaboración propia.

ESCENARIO : TRIGGERS ASINCRONOS		
PRUEBA 1		PRUEBA 2
PROCESOS	BLOQUEOS 1 %	BLOQUEOS 2 %
1000	5.15	5.01
1500	4.2	4.12
1800	3.18	3.09
2000	4.22	4.85
5000	3.60	3.51
10000	4.02	4.08

Tabla 23

Cantidad de procesos y bloqueos expresados en porcentaje, en un escenario con triggers asíncronos.

Fuente: Elaboración propia.

ESCENARIO : TRIGGERS ASINCRONOS				
PRUEBA 1			PRUEBA 2	
PROCESOS	BLOQUEOS 1 %	CALIFICACION	BLOQUEOS 2 %	CALIFICACION
1000	5.15	5	5.01	5
1500	4.2	5	4.12	5
1800	3.18	5	3.09	5
2000	4.22	5	4.85	5
5000	3.60	5	3.51	5
10000	4.02	5	4.08	5

Tabla 24

Tabla de casos de concurrencia, entorno a tiempos, para los casos de prueba 1 y 2.

Fuente: Elaboración propia.

ESCENARIO :	TRIGGERS ASINCRONOS					
	CALIFICACION					
BLOQUEOS 1 %	5	5	5	5	5	5
BLOQUEOS 2 %	5	5	5	5	5	5

Tabla 25

Reordenación de la tabla de concurrencia entorno a las demás calificaciones.

Fuente: Elaboración propia.

	1	2	3	4	5	Σ
1	0	0	0	0	0	**0**
2	0	0	0	0	0	**0**
3	0	0	0	0	0	**0**
4	0	0	0	0	0	**0**
5	0	0	0	0	6	**1**
Σ	**0**	**0**	**0**	**0**	**0**	

Cálculo de Po

Po=1

Cálculo de Pe

Pe=0

Cálculo coeficiente Kappa

k=1

La taza de operaciones por frecuencia de bloqueo, es un cálculo realizado a partir de la división total de los bloqueos por la cantidad de operaciones realizadas.

Por otro lado podemos ver, que la tasa de crecimiento de una replicación síncrona crece aceleradamente, cuándo la cantidad de procesos también crece.

Todo esto debido al primer paso propuesto, en el esquema algoritmico, ya que alli se realiza un respaldo de los datos a replicarse, y en caso de error, simplemente retornamos los datos no replicados, para replicarlos de nuevo.

Por lo tanto, analizando cada una de los resultados para el coeficiente kappa, asignados a un determinado escenario, podemos ver que el escenario asincrono es el mejor entre los tres, ya que muestra coherencia al 100% en Po, debido a que los porcentajes de error rebajaron , y esto hace que todos los casos pertenezcan a una calificacion tipo .

El coeficiente kappa para un escenario asincrono es 1, osea 100%, por lo tanto podemos decir que la variable 2 bloqueos es verdadera, por lo tanto en un método asincrono, en promedio existe un porcentaje de bloqueo menor al 3%, muchisimo menor al que genera el método síncrono.

4.1.3 Variable 3: Rendimiento

La actualización usando un método de replicación asíncrona, tiene un rendimiento, medido por el número de operaciones que se actualiza por segundo, mayor que cuando se utiliza el método sincrónico.

Ahora analizaremos el coeficiente kappa, entorno a la cantidad de operaciones, para cada los escenarios síncronos y asíncronos, y los tipos de rendimiento bajo los parámetros explicados en la tabla 1 , donde 5 es un buen tiempo y 1 es el peor tiempo en operaciones de replicación expresado en segundos.

.

4.1.3.1 Análisis del coeficiente Kappa en un escenario utilizando triggers síncronos y un tipo de rendimiento bajo medio.

Tabla 26

Cantidad de operaciones y tiempo expresado en segundos, en un escenario con triggers síncronos.

Fuente: Elaboración propia.

ESCENARIO : TRIGGERS SINCRONOS		
TIPO DE RENDIMIENTO : BAJO Y MEDIO		
PRUEBA 1		PRUEBA 2
OPERACIONES	TIEMPO 1	TIEMPO 2
10	35.41	33.54
30	55.10	58.45
50	58.23	60.12
100	60.41	59.85
200	55.12	55.41
500	50.41	52.14

Tabla 27

Cantidad de operaciones y tiempo expresado en segundos, en un escenario con triggers asíncronos.

Fuente: Elaboración propia.

ESCENARIO : TRIGGERS SINCRONOS				
TIPO DE RENDIMIENTO : BAJO Y MEDIO				
PRUEBA 1			PRUEBA 2	
OPERACIONES	TIEMPO 1	CALIFICACION	TIEMPO 2	CALIFICACION
10	35.41	4	33.54	4
30	55.10	3	58.45	3
50	58.23	3	60.12	2
100	60.41	2	59.85	3
200	55.12	3	55.41	3
500	50.41	3	52.14	3

Tabla 28

Tabla de casos de concurrencia, entorno a tiempos, para los casos de prueba 1 y 2.

Fuente: Elaboración propia.

ESCENARIO :	TRIGGERS SINCRONOS					
	CALIFICACION					
TIEMPO 1	4	3	3	2	3	3
TIEMPO 2	4	3	2	3	3	3

Tabla 29

Reordenación de la tabla de concurrencia entorno a las demás calificaciones.

Fuente: Elaboración propia.

	1	2	3	4	5	Σ
1	0	0	0	0	0	0
2	0	0	1	0	0	0.17
3	0	1	3	0	0	0.67
4	0	0	0	1	0	0.17
5	0	0	0	0	0	0
Σ	0	0.17	0.67	0.17	0	

Cálculo de Po

Po=0.67

Cálculo de Pe

Pe=0.5

Cálculo coeficiente Kappa

k=0.33

4.1.3.2 Análisis del coeficiente Kappa en un escenario utilizando triggers síncronos y un tipo de rendimiento alto.

Tabla 30

Cantidad de operaciones y tiempo expresado en segundos, en un escenario con triggers síncronos.

Fuente: Elaboración propia.

ESCENARIO : TRIGGERS SINCRONOS		
TIPO DE RENDIMIENTO : ALTO		
PRUEBA 1		PRUEBA 2
OPERACIONES	TIEMPO 1	TIEMPO 2
1000	59.21	61.23
3000	65.25	63.21
5000	66.21	66.32
10000	68.41	68.54
15000	75.25	72.56
20000	85.51	80.52

Tabla 31

Cantidad de operaciones y tiempo expresado en segundos, en un escenario con triggers asíncronos.

Fuente: Elaboración propia.

ESCENARIO : TRIGGERS SINCRONOS				
TIPO DE RENDIMIENTO : ALTO				
PRUEBA 1			PRUEBA 2	
OPERACIONES	TIEMPO 1	CALIFICACION	TIEMPO 2	CALIFICACION
1000	59.21	3	61.23	2
3000	65.25	2	63.21	2
5000	66.21	2	66.32	2
10000	68.41	2	68.54	2
15000	75.25	2	72.56	2
20000	85.51	1	80.52	1

Tabla 32

Tabla de casos de concurrencia, entorno a tiempos, para los casos de prueba 1 y 2.

Fuente: Elaboración propia.

ESCENARIO :	TRIGGERS SINCRONOS					
	CALIFICACION					
TIEMPO 1	3	2	2	2	2	1
TIEMPO 2	2	2	2	2	2	1

Tabla 33

Reordenación de la tabla de concurrencia entorno a las demás calificaciones.

Fuente: Elaboración propia.

	1	2	3	4	5	Σ
1	1	0	0	0	0	0.17
2	0	4	1	0	0	0.83
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
Σ	0.166666666667	0.67	0.17	0	0.17	

Cálculo de Po

Po=0.83

Cálculo de Pe

Pe=0.58

Cálculo coeficiente Kappa

k=0.6

4.1.3.3 Análisis del coeficiente Kappa en un escenario utilizando triggers asíncronos y un tipo de rendimiento bajo medio.

Tabla 34

Cantidad de operaciones y tiempo expresado en segundos, en un escenario con triggers asíncronos.

Fuente: Elaboración propia.

ESCENARIO : TRIGGERS ASINCRONOS		
TIPO DE RENDIMIENTO : BAJO Y MEDIO		
PRUEBA 1		PRUEBA 2
OPERACIONES	TIEMPO 1	TIEMPO 2
10	59.85	64.52
30	65.52	65.85
50	62.45	61.98
100	61.52	61.36
200	59.84	60.85
500	48.85	48.85

Tabla 35

Cantidad de operaciones y tiempo expresado en segundos, en un escenario con triggers asíncronos.

ESCENARIO : TRIGGERS ASINCRONOS				
TIPO DE RENDIMIENTO : BAJO Y MEDIO				
PRUEBA 1			PRUEBA 2	
OPERACIONES	TIEMPO 1	CALIFICACION	TIEMPO 2	CALIFICACION
10	59.85	3	64.52	2
30	65.52	2	65.85	2
50	62.45	2	61.98	2
100	61.52	2	61.36	2
200	59.84	3	60.85	2
500	48.85	3	48.85	3

Fuente: Elaboración propia.

Tabla 36

Tabla de casos de concurrencia, entorno a tiempos, para los casos de prueba 1 y 2.

Fuente: Elaboración propia.

ESCENARIO :	TRIGGERS ASINCRONOS					
	CALIFICACION					
TIEMPO 1	3	2	2	2	3	3
TIEMPO 2	2	2	2	2	2	3

Tabla 37

Reordenación de la tabla de concurrencia entorno a las demás calificaciones.

Fuente: Elaboración propia.

	1	2	3	4	5	Σ
1	0	0	0	0	0	**0**
2	0	3	2	0	0	**0.83**
3	0	0	1	0	0	**0.17**
4	0	0	0	0	0	**0**
5	0	0	0	0	0	**0**
Σ	**0**	**0.5**	**0.5**	**0**	**0**	

Cálculo de Po

Po=0.67

Cálculo de Pe

Pe=0.5

Cálculo coeficiente Kappa

k=0.33

4.1.3.4 Análisis del coeficiente Kappa en un escenario utilizando triggers asíncronos y un tipo de rendimiento alto.

Tabla 38

Cantidad de operaciones y tiempo expresado en segundos, en un escenario con triggers asíncronos.

Fuente: Elaboración propia.

ESCENARIO : TRIGGERS ASINCRONOS		
TIPO DE RENDIMIENTO : ALTO		
PRUEBA 1		PRUEBA 2
OPERACIONES	TIEMPO 1	TIEMPO 2
1000	55.95	55.85
3000	55.10	56.10
5000	55.02	54.98
10000	59.41	59.65
15000	60.58	60.75
20000	61.59	61.75

Tabla 39

Cantidad de operaciones y tiempo expresado en segundos, en un escenario con triggers asíncronos.

ESCENARIO : TRIGGERS ASINCRONOS				
TIPO DE RENDIMIENTO : ALTO				
PRUEBA 1			PRUEBA 2	
OPERACIONES	TIEMPO 1	CALIFICACION	TIEMPO 2	CALIFICACION
1000	55.95	3	55.85	3
3000	55.10	3	56.10	3
5000	55.02	3	54.98	3
10000	59.41	3	59.65	3
15000	60.58	2	60.75	2
20000	61.59	2	61.75	2

Fuente: Elaboración propia.

Tabla 40

Tabla de casos de concurrencia, entorno a tiempos, para los casos de prueba 1 y 2.

Fuente: Elaboración propia.

ESCENARIO :	TRIGGERS ASINCRONOS					
	CALIFICACION					
TIEMPO 1	3	3	3	3	2	2
TIEMPO 2	3	3	3	3	2	2

Tabla 41

Reordenación de la tabla de concurrencia entorno a las demás calificaciones.

Fuente: Elaboración propia.

	1	2	3	4	5	Σ
1	0	0	0	0	0	**0**
2	0	2	0	0	0	**0.33**
3	0	0	4	0	0	**0.67**
4	0	0	0	0	0	**0**
5	0	0	0	0	0	**0**
Σ	**0**	**0.33**	**0.67**	**0**	**0**	

Cálculo de Po

Po=1

Cálculo de Pe

Pe=0.56

Cálculo coeficiente Kappa

k=1

Por lo tanto, analizando cada una de los resultados para el coeficiente kappa, asignados a un determinado escenario, podemos ver que el escenario síncrono es el mejor entre los dos, en ambientes donde el rendimiento es bajo .

Un ambiente con rendimiento bajo, es un ambiente irreal, donde no exista una concurrencia alta. Podemos ver algo muy curioso, e interesante, el desempeño cuando las cantidades de operaciones son 10 y 50, es mucho más grande, que cuando existe concurrencia.

Por otro lado cuando la cantidad de operaciones es 100, el tiempo en segundos, se duplica, y cuando la cantidad de operaciones llega a 150 y 200 , el tiempo va reduciendo.

Curiosamente, el método síncrono siempre gana cuando no hay concurrencia.

Esto se puede justificar, porque el método asincrónico debe registrar todas las operaciones en forma temporal para que, más tarde, el proceso de replicación pueda actualizar la tabla de destino.

El coeficiente kappa para un escenario asíncrono es 1, osea 100%, por lo tanto podemos decir que la variable 3 rendimiento es verdadera, porque en un ambiente con alto rendimiento, el metodo asíncrono es el mejor, ya que los tiempos van reduciendo, conforme las operaciones van aumentando.

CAPÍTULO V:

CONCLUSIONES Y RECOMENDACIONES

Finalmente, se describe las conclusiones y recomendaciones, acerca de la investigación, con el único fin, de motivar a estudiantes a que realicen investigaciones futuras acerca del tema.

5.1 Conclusiones

En esta sección la hipótesis del experimento será resuelta, con el apoyo de los análisis anteriores.

Partiendo de la hipótesis principal, y haciendo referencia a las variables de apoyo, analizadas en el apartado 2.4.2.1, tenemos las siguientes conclusiones

Variable 1: Tiempo. El tiempo para procesar un método de operación asincrónica pendiente es pequeña y está en el orden de decenas de milisegundos.

La variable 1 fue demostrada en el capítulo anterior, apartado 4.1.1, donde se realizó un análisis mediante el método estadístico coeficiente kappa, entorno a la cantidad de procesos y los tiempos de procesamiento para cada nodo de replicación, donde se concluyó, que el tiempo promedio en procesamientos de replicación varió de 4 milisegundos a 30 milisegundos para todos los niveles de concurrencia, es decir casi un 50% menor al método síncrono.

Variable 2: Bloqueos.- El Método síncrono genera muchos bloqueos y la cantidad es mayor que el método asincrónico.

La variable 2 también fue demostrada en el capítulo anterior, apartado 4.1.2, donde se realizó un análisis mediante el método estadístico coeficiente kappa, entorno a la cantidad de procesos y la cantidad de bloqueos expresados en porcentaje, para cada nodo de replicación, donde se demuestra que la variable es verdadera.

En todos los niveles de concurrencia, el método sincrónico genera por lo menos tres veces más obstrucciones que un escenario sin trigger y por lo menos el doble que el método asincrónico.

Variable 3: Rendimiento.-. La actualización usando un método de replicación asíncrona, tiene un rendimiento, medido por el número de operaciones que se actualiza por segundo, mayor que cuando se utiliza el método sincrónico.

Por último ,la variable 3 también fue demostrada en el capítulo anterior, apartado 4.1.3, donde se realizó un análisis mediante el método estadístico coeficiente kappa, entorno a la cantidad de operaciones y los tiempos de procesamiento para cada nodo de replicación,

respecto a un rendimiento alto, medio y bajo, donde se demuestra que la variable es verdadera.

Ya que se demostró que para los medios y altos niveles de concurrencia, el método asíncrono es el mejor. La excepción es cuando el nivel de concurrencia es bajo, en el que el método sincrónico tiene un mejor rendimiento debido a la cantidad de bloqueos no interfieren con su rendimiento.

En esta situación, no hay ninguna ventaja en el trabajo de forma asincrónica. Es importante señalar que muchos sistemas no críticos trabajan en una baja concurrencia y por lo tanto la aplicabilidad de la solución síncrona no es alta.

Por otra parte, para los sistemas críticos típicos, que tienen un nivel medio o alto de la concurrencia, la mejor solución es el método asíncrono para la replicación de datos.

El rendimiento es un factor clave en esta situación y las ganancias de 100% con respecto al método síncrono son notorias, ya que si trabajamos con un método síncrono bajo una alta concurrencia de datos, existirán problemas de tiempo insuficiente o tiempo de espera excedido o limitado.

Entorno a los objetivos específicos, podemos concluir:

Los objetivos específicos 1,2 y 3 , fueron demostrados bajo el análisis de las variables 1, 2 y 3, ya que la utilización de métodos asíncronos en los procesos de replicación ,entorno a la estructura algorítmica propuesta, no genera bloqueos, ofrece un mejor rendimiento y los tiempos promedios de proceso son menores al síncrono.

Por todo lo mencionado anteriormente, en relación a los parámetros de experimentación entorno a las variables de apoyo, la hipótesis principal de esta investigación queda entonces pues demostrada.

5.2 Recomendaciones

Mediante el presente tópico deseo motivar a otros estudiantes de pre y porque no, de postgrado, a realizar investigaciones similares, acerca del tema, ya que a mi parecer solo se investigó, solo uno de los muchos factores que podrían mejorar la lógica de replicación de bases de datos.

A continuación, se describe algunos temas de investigación, que a mi parecer son importantes, para investigaciones futuras:

El esquema algorítmico se puede mejorar, tomando en cuenta nuevos coeficientes de variabilidad, los cuales pueden ser generados y almacenados conforme se hagan las mediciones periódicas de las fuentes emisoras más significativas en nuestro entorno de trabajo.

Tras la aplicación de la herramienta, es importante poner en su uso en un entorno real con una base de datos utilizada por un gran número de aplicaciones distintas y heterogéneas. En este entorno, es necesario volver a hacer el experimento para verificar el comportamiento de la replicación asincrónica. Aunque el rendimiento de la solución es un factor relevante para ser medido y verificado, es posible que, en un entorno real, definimos métricas adicionales relacionados con la usabilidad que nos informan si realmente, con la herramienta, las refactorizaciones base de datos se pueden implementar de una manera fácil e intuitiva.

Alcanzando este punto en el que se ha cumplido con el último paso (Conclusiones y recomendaciones), la presente investigación llega a su término.

REFERENCIAS BIBLIOGRÁFICAS

6.1 Física.

[AMB 06]

AMBLER SCOTT & SADALAGE PRAMOD. Refactoring Databases: Evolutionary Database Design. Chicago,Estados Unidos.2006

[BER 11]

BERTONE RODOLFO. Artículo: Análisis de rendimiento y replicación en Bases de Datos distribuidas. Universidad de La Plata. La Plata, Argentina.2011.

[CAS 00]

CASTILLO VALDIVIESO PEDRO A. .Tesis de doctorado: Optimización de perceptrones multicapa mediante algoritmos evolutivos (A.E). Asociación Española para la Inteligencia Artificial. Alicante. España.2000.

[DAR 09]

DARWIN & WALLACE. Editorial CSIC. Selección natural: tres fragmentos para la historia. Universidad Nacional Autónoma de México. Ciudad de México, Distrito Federal. México.2009

[GAV 02]

GAVILLAUD JEROME. Editorial ENI . SQL y Algebra relacional. Barcelona. España.2002

[JUA 11]

JUAREZ RODRIGUEZ JOSÉ RAMÓN. Tesis doctoral: Protocolos informáticos de replicación de bases de datos. Universidad Pública de Navarra. Madrid, España.2011.

[JAI 91]

JAIN RAJ.The Art Of Computer Systems Performance Analysis. Nueva York, Estados Unidos.1991.

[MIL 08]

MILÁN FRANCO JESÚS MANUEL. Tesis doctoral: Replicación Autonómica de Bases de Datos. Universidad Politécnica de Madrid. Madrid, España.2008.

[MON 05]

MONROY HERNÁNDEZ RENÉ AMILCAR. Tesis de grado: Lenguaje xml como solución a las bases de datos y su replicación. Universidad de San Carlos de Guatemala.Guatemala.2005.

[SIL 02]

SILBERSCHATZ ABRAHAM. Editorial Mc. Graw Hill . Fundamentos de Base de datos .Nueva York. Estados Unidos.2002.

[TOZ 09]

TOZO DE CARBALLO GUILLERMO. Tesis de Maestría: Aplicación de prácticas de agentes en la construcción de Data Warehouse Evolutivo. Instituto de matemática y estadística USPI. Sao Paolo – Brasil.2009.

[WAL 09]

WALLACE $ DARWIN . Editorial Critica. La teoría de la evolución de las especies. España.2009.

6.2 Recursos web

[ALV 12]

Álvarez Humberto (9 de marzo de 2012).”Oracle Golden Gate - Flujo de Replicación Initial Load ”Recuperado el 22 de mayo de 2014 de http://simpleintegrador.blogspot.com/2012/03/oracle-goldengate-flujo-de-replicacion.html

[BUR 11]

BURMESTER JONATHAN (27 de septiembre de 2011).”Técnicas de fragmentación, replicación y reparto de datos para el diseño de base de datos distribuidas”. Recuperado el 13 de mayo de 2014 de http://bdblog5.blogspot.com/2011/09/tecnicas-de-fragmentacion-replicacion-y.html

[BNY 13]

BENEITO RAUL. (2013, 10 de enero). ¿Cuánta Información se Genera y Almacena en el Mundo? de: http://documania20.wordpress.com/2013/09/16/cuanta-informacion-se-genera-y-almacena-en-el-mundo/. Recuperado el 10 de mayo de 2013

[CON 11]

CONTINUENT.(2011, 15 de febrero).Bristlecone performance test de http://www.continuent.com/. Recuperado el 24 de julio de 2013.

[FER 2000]

FERRERAS FERRERAS, JUAN CARLOS (2000). “Visualización interactiva del algoritmo de Fortune en internet”, Trabajo de fin de carrera, Universidad Politécnica de Madrid. Departamento de Matemática Aplicada, Facultad de Informática. http://www.dma.fi.upm.es/mabellanas/tfcs/flips/Intercambios/html/teoria/teoria_del_esp.htm#RepresTridimensional. Recuperado el 31 de septiembre de 2013.

[FIG 11]

FAJARDO CARMINA , FIGUEROA CLAUDIA, FIGUEROA JEIMHY. (29 de enero de 2011). ¨Replicación de bases de datos Oracle¨. Recuperado el 11 de mayo de 2014 de http://basesdedatosues.blogspot.com/2010/07/replicacion-oracle-streams.htm

[PER 09]

PÉREZ MATA MANEL (26 de mayo de 2009).”Replicación asíncrona unidireccional Maestro”. Recuperado el 21 de mayo de 2014 de http://manelperez.com/bases-de-datos/replicacion-asincrona-unidireccional-maestro-esclavo-en-mysql-50/

[STA 2013]

STAFFORD STEVE. (2013, 21 de noviembre). Room Area Net vs Gross de: http://revitoped.blogspot.com/2013/11/room-area-net-vs-gross.html. Recuperado el 21 de mayo de 2013

[SYB 13]
Sybven, Corporación (05 de Abril de 2013). “Replicación de datos ” . Recuperado el 10 de mayo de 2014, de http://www.corporacionsybven.com/portal/index.php.

ANEXOS

7.1 ANEXO A

CONFIGURACIÓN PARA LA BASE DE DATOS POSTGRESQL

```
# -----------------------------
# PostgreSQL configuration file
# -----------------------------
# Memory units:  kB = kilobytes        Time units:  ms  = milliseconds
#                MB = megabytes                     s   = seconds
#                GB = gigabytes                     min = minutes
#                                                   h   = hours
#                                                   d   = days


#------------------------------------------------------------------------------
# FILE LOCATIONS
#------------------------------------------------------------------------------
data_directory = '/var/lib/postgresql/8.4/main'         # use data in another directory
hba_file = '/etc/postgresql/8.4/main/pg_hba.conf'       # host-based authentication file
ident_file = '/etc/postgresql/8.4/main/pg_ident.conf'   # ident configuration file
external_pid_file = '/var/run/postgresql/8.4-main.pid'          # write an extra PID file

#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------
listen_addresses = '*'          # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost', '*' = all
                                        # (change requires restart)
port = 5432
max_connections = 200
unix_socket_directory = '/var/run/postgresql'
ssl = true

#------------------------------------------------------------------------------
```

7.2 ANEXO B

GUÍA DE USUARIO

- INSTALACIÓN DEL SERVIDOR DE BASE DE DATOS POSTGRESQL

-----PREPARANDO EL ESCENARIO --------

```
sudo apt-get update
sudo apt-get install postgresql
sudo nano /etc/postgresql/9.3/main/pg_hba.conf
sudo nano /etc/postgresql/9.3/main/postgresql.conf
sudo /etc/init.d/postgresql restart
sudo su - postgres

psql -U postgres

ALTER USER postgres WITH PASSWORD '$PASSWORD';
```

- ACCEDER AL SERVIDOR VIA SSH
- INSTALAR BUCARDO

---------INSTALAR BUCARDO-------

```
wget http://bucardo.org/downloads/dbix_safe.tar.gz
tar xvfz dbix_safe.tar.gz
cd DBIx-Safe-1.2.5
sudo apt-get install make
 perl Makefile.PL
 make
 sudo make install
wget http://bucardo.org/downloads/Bucardo-5.2.0.tar.gz
 tar xvfz Bucardo-5.2.0.tar.gz
 cd Bucardo-5.2.0
 perl Makefile.PL
 make
 sudo make install
apt-get install postgresql-plperl-9.3
sudo aptitude install libdbd-pg-perl
```

- CONFIGURAR BUCARDO

```
bucardo install --piddir=/tmp
------------------AÑADIENDO LAS BASES DE DATOS A REPLICARSE-------------
---
bucardo add database server1 dbname=bdtesis
bucardo add database server2 dbname=bdtesis host=54.149.62.102
user=bucardo password=mija16291 port=5432
```

```
------------------AÑADIENDO LAS TABLAS A REPLICARSE----------------
bucardo add table cambio db=server1
bucardo add table cambio db=server2
------------------AÑADIENDO GRUPOS DE REPLICACION-----------------------------
------
bucardo add herd server-a_server-b cambio db=server1
bucardo add herd server-b_server-a cambio db=server2
-----------------DEFINIENDO ORIGEN Y DESTINO-------------------------------------
bucardo add sync bdtesis_sync_a relgroup=server-a_server-b
dbs=server1,server2
bucardo add sync bdtesis_sync_b relgroup=server-b_server-a
dbs=server2,server1
-------------------EJECUTANDO-----------------------------------------------
bucardo start
```

7.3 ANEXO C

ESTRUCTURA DE CÓDIGO EN PHP, PARA GENERAR EL ANÁLISIS DEL COEFICIENTE KAPPA AUTOMATICAMENTE

```
<?php
function redondear_dos_decimal($valor) {
$float_redondeado=round($valor * 100) / 100;
return $float_redondeado;
}
?>
<tr>
<td class="Estilo1"><div align="center"><strong>1</strong></div></td>
<td class="Estilo1"><div align="center"><?php echo $a=0+ $_POST['a'];?></div></td>
<td class="Estilo1"><div align="center"><?php echo $b=0+ $_POST['b'];?></div></td>
<td class="Estilo1"><div align="center"><?php echo $c=0+ $_POST['c'];?></div></td>
<td class="Estilo1"><div align="center"><?php echo $d=0+ $_POST['d'];?></div></td>
<td class="Estilo1"><div align="center"><?php echo $e=0+ $_POST['e'];?></div></td>
<td class="Estilo1"><div align="center"><strong><?php echo redondear_dos_decimal($suma1=0+($a+$b+$c+$d+$e)/6);?></strong></div></td>
</tr>
<tr>
<td class="Estilo1"><div align="center"><strong>2</strong></div></td>
<td class="Estilo1"><div align="center"><?php echo $f=0+ $_POST['f'];?></div></td>
<td class="Estilo1"><div align="center"><?php echo $g=0+ $_POST['g'];?></div></td>
<td class="Estilo1"><div align="center"><?php echo $h=0+ $_POST['h'];?></div></td>
<td class="Estilo1"><div align="center"><?php echo $i=0+ $_POST['i'];?></div></td>
<td class="Estilo1"><div align="center"><?php echo $j=0+ $_POST['j'];?></div></td>
<td class="Estilo1"><div align="center"><strong><?php echo redondear_dos_decimal($suma2=0+($f+$g+$h+$i+$j)/6);?></strong></div></td>
</tr>
<tr>
<td class="Estilo1"><div align="center"><strong>3</strong></div></td>
```

<td class="Estilo1"> <?php echo $k=0+ $_POST['k'];?> </td>

<td class="Estilo1"> <?php echo $l=0+ $_POST['l'];?> </td>

<td class="Estilo1"> <?php echo $m=0+ $_POST['m'];?> </td>

<td class="Estilo1"> <?php echo $n=0+ $_POST['n'];?> </td>

<td class="Estilo1"> <?php echo $o=0+ $_POST['o'];?> </td>

<td class="Estilo1"> <?php echo redondear_dos_decimal($suma3=0+ ($k+$l+$m+$n+$o)/6);?> </td>

</tr>

<tr>

<td class="Estilo1"> 4 </td>

<td class="Estilo1"> <?php echo $p=0+ $_POST['p'];?> </td>

<td class="Estilo1"> <?php echo $q=0+ $_POST['q'];?> </td>

<td class="Estilo1"> <?php echo $r=0+ $_POST['r'];?> </td>

<td class="Estilo1"> <?php echo $s=0+ $_POST['s'];?> </td>

<td class="Estilo1"> <?php echo $t=0+ $_POST['t'];?> </td>

<td class="Estilo1"> <?php echo redondear_dos_decimal($suma4=0+ ($p+$q+$r+$s+$t)/6);?> </td>

</tr>

<tr>

<td class="Estilo1"> 5 </td>

<td class="Estilo1"> <?php echo $u=0+ $_POST['u'];?> </td>

<td class="Estilo1"> <?php echo $v=0+ $_POST['v'];?> </td>

<td class="Estilo1"> <?php echo $w=0+ $_POST['w'];?> </td>

<td class="Estilo1"> <?php echo $x=0+ $_POST['x'];?> </td>

<td class="Estilo1"> <?php echo $y=0+ $_POST['y'];?> </td>

<td class="Estilo1"> <?php echo redondear_dos_decimal($suma5=0+ ($u+$v+$w+$x+$y)/6);?> </td>

</tr>

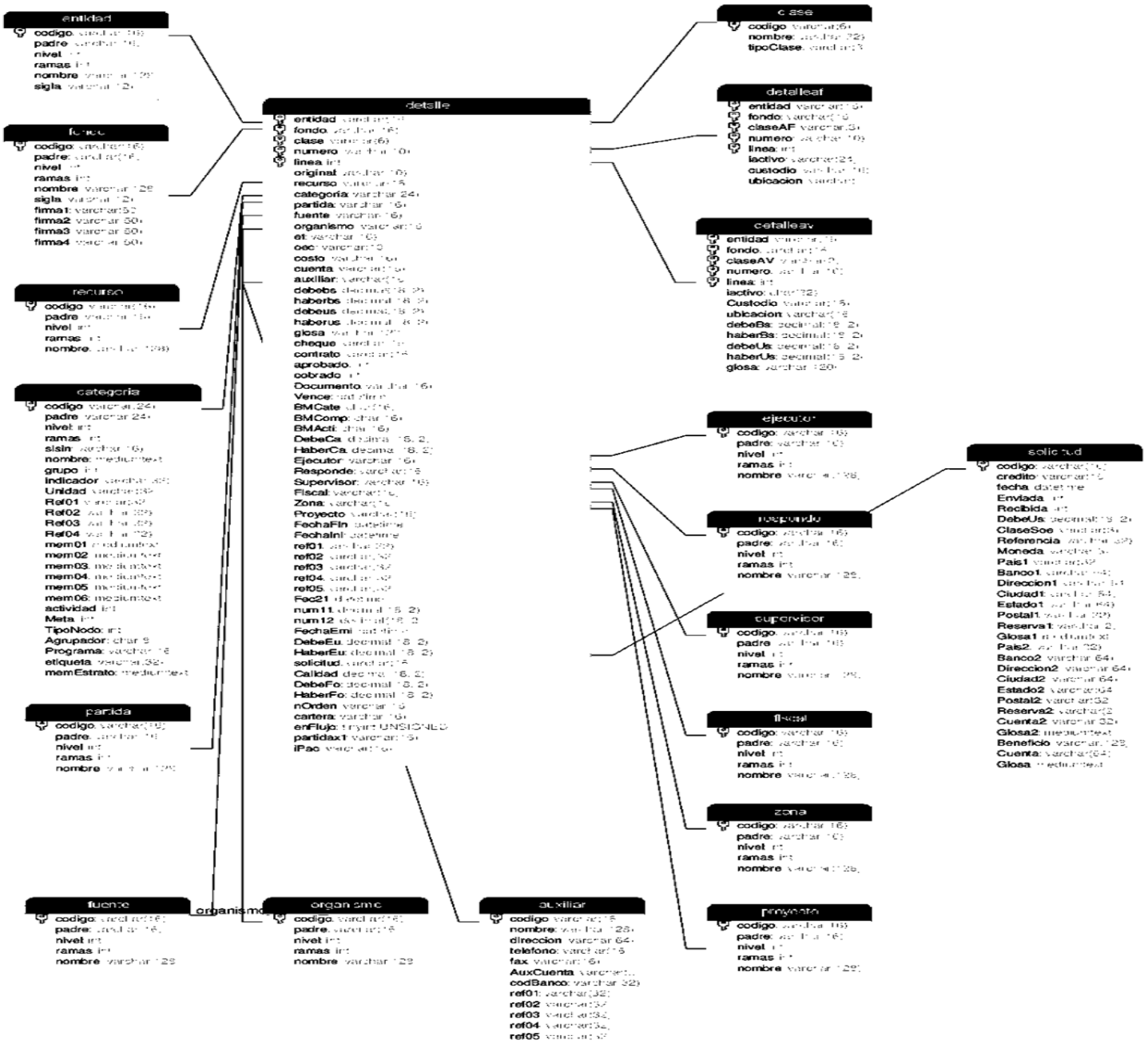
<tr>

<td class="Estilo4"> <h1 align="center" id="firstHeading" lang="es" xml:lang="es">Σ</h1></td><td class="Estilo1"> <?php echo $sumav1=($a+$f+$k+$p+$u)/6; ?> </td>

```
<td class="Estilo1"><div align="center"><strong><?php echo redondear_dos_decimal($sumav2=($v+$g+$l+$q+$v)/6); ?></strong></div></td>

<td class="Estilo1"><div align="center"><strong><?php echo redondear_dos_decimal($sumav3=($c+$h+$r+$m+$w)/6); ?></strong></div></td>

<td class="Estilo1"><div align="center"><strong><?php echo redondear_dos_decimal($sumav4=($d+$i+$n+$s+$x)/6); ?></strong></div></td>

<td class="Estilo1"><div align="center"><strong><?php echo redondear_dos_decimal($sumav5=($a+$f+$k+$p+$u)/6); ?></strong></div></td>

<td class="Estilo1"><div align="center"></div></td>

</tr>

</table>

<span class="Estilo1">

<?php $repetidos=$_POST['repetidos'];?>

</span>

<p class="Estilo2">C&aacute;lculo de Po</p>

<p class="Estilo1">Po=<?php echo redondear_dos_decimal($po=(($repetidos)/($repetidos+(6-$repetidos))));?></p>


<p class="Estilo1"><span class="Estilo2">C&aacute;lculo de Pe</span></p>

<p class="Estilo1">Pe=<?php echo redondear_dos_decimal($pe=(($suma1*$sumav1)+($suma2*$sumav2)+($suma3*$sumav3)+($suma4*$sumav4)+($suma5*$sumav5)));?></p>

<p class="Estilo1"><span class="Estilo2">C&aacute;lculo coeficiente Kappa </span></p>

<p class="Estilo1">k=<?php echo redondear_dos_decimal($resultado=(($po-$pe)/(1-$pe)));?></p>
```

7.4 ANEXO D

ESTRUCTURA DE LA BASE DE DATIOS AUXILIAR


entidad
codigo
padre
nivel
ramas
nombre
sigla
fondo
codigo
padre
nivel
ramas
nombre
sigla
firma1
firma2
firma3
firma4
recurso
codigo
padre
nivel
ramas
nombre
categoria
codigo
padre
nivel
ramas
sisIn
nombre
grupo
Indicador
Unidad
Ref01
Ref02
Ref03
Ref04
mem01
mem02
mem03
mem04
mem05
mem06
actividad
Meta
TipoNodo
Agrupador
Programa
etiqueta
memEstrato
partida
codigo
padre
nivel
ramas
nombre
fuente
codigo
padre
nivel
ramas
nombre
organismo
codigo
padre
nivel
ramas
nombre
detalle
entidad
fondo
clase
numero
linea
original
recurso
categoria
partida
fuente
organismo
et
oec
costo
cuenta
auxiliar
debebs
haberbs
debeus
haberus
glosa
cheque
contrato
aprobado
cobrado
Documento
Vence
BMCate
BMComp
BMActi
DebeCa
HaberCa
Ejecutor
Responde
Supervisor
Fiscal
Zona
Proyecto
FechaFin
FechaIni
ref01
ref02
ref03
ref04
ref05
Fec21
num11
num12
FechaEmi
DebeEu
HaberEu
solicitud
Calidad
DebeFo
HaberFo
nOrden
cartera
enFlujo
partidax1
iPac
auxiliar
codigo
nombre
direccion
telefono
fax
AuxCuenta
codBanco
ref01
ref02
ref03
ref04
ref05
clase
codigo
nombre
tipoClase
detalleaf
entidad
fondo
claseAF
numero
linea
iactivo
custodio
ubicacion
detalleav
entidad
fondo
claseAV
numero
linea
iactivo
Custodio
ubicacion
debeBs
haberBs
debeUs
haberUs
glosa
ejecutor
codigo
padre
nivel
ramas
nombre
responde
codigo
padre
nivel
ramas
nombre
supervisor
codigo
padre
nivel
ramas
nombre
fiscal
codigo
padre
nivel
ramas
nombre
zona
codigo
padre
nivel
ramas
nombre
proyecto
codigo
padre
nivel
ramas
nombre
solicitud
codigo
credito
fecha
Enviada
Recibida
DebeUs
ClaseSoe
Referencia
Moneda
Pais1
Banco1
Direccion1
Ciudad1
Estado1
Postal1
Reserva1
Glosa1
Pais2
Banco2
Direccion2
Ciudad2
Estado2
Postal2
Reserva2
Cuenta2
Glosa2
Beneficio
Cuenta
Glosa