Universidad de las Ciencias Informáticas.

Facultad 4

Título: Disponibilidad y accesibilidad a datos
en la plataforma Génesis.

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas.

Autores: Yadira Lizama Mué
Mario Michel Concepción Milanés

Tutor: Lic. Edisel Navas Conyedo

Ciudad de la Habana, Mayo 2009

Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 4 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los _____ días del mes de _____ del año _____ .

Autores:
Yadira Lizama Mué

Tutor:
Lic. Edisel Navas Conyedo

_____

Firma del Autor

_____

Firma del Tutor.

Mario Michel Concepción Milanés

_____

Firma del Autor

Agradecimientos

A mis padres por el amor y la fe en mi.

A Lissuán por estar siempre a mi lado.

A la Revolución Cubana y a Fidel por darme la oportunidad de estudiar en la Universidad del Futuro.

A EIPAD por enseñarme el verdadero valor de la unión, el trabajo y la amistad. Por enseñarme que los sueños se cumplen.

A mi familia, mis suegros y mis vecinos por el ánimo y la confianza en mí.

A todos mis amigos, los de toda la vida; a quienes les debo gran parte de mi formación como profesional y como ser humano.

A los colaboradores cubanos y venezolanos en la Misión Medica Cubana en Venezuela en el CDI Tariba Estado Táchira (Marzo/2008- Agosto/2008) por quererme como una hija más.

A la Vida.

Yadira Lizama Mué.

A Fidel por permitirme ser parte de este sueño hecho realidad que es la UCI, gracias por ser nuestro faro y guía.

A todos los que aportaron su granito de arena en mi formación.

A EIPAD por mostrarme que nada es imposible.

A todos mis amigos y hermanos.

A todos, simplemente Gracias!

Mario Michel Concepción Milanés.

Dedicatoria

A mis padres. A Lissuán. A mi familia y amigos. A EIPAD.

Yadira Lizama Mué.

A mis padres por su confianza y apoyo incondicional.

A la memoria de mi abuela que siempre soñó verme convertido en Ingeniero.

A mi novia por su amor, su comprensión y apoyo.

A mis hermanos de la UCI que nunca olvidaré.

A mi familia.

Mario Michel Concepción Milanés.

Resumen

Los procesos de negocio involucran cada día mayores volúmenes de información y la complejidad de sus relaciones evoluciona de manera creciente por lo que las empresas e instituciones científicas apuestan por mejorar o crear nuevas alternativas eficientes y menos costosas para su modelación. La adquisición de superordenadores para procesar cantidades de datos considerables, no es una alternativa permisible para países subdesarrollados como Cuba debido a su alto costo, el cambio constante de tecnología y las sumas en inversión en soporte y mantenimiento que traen asociados. La computación paralela se presenta como una solución factible para el procesamiento de grandes volúmenes de datos con el empleo de pocos recursos, con gran aplicación para la construcción de Sistemas de Toma de Desiciones en Tiempo Real, desarrollo de Técnicas de Inteligencia Empresarial y Minerías de Datos. Las múltiples arquitecturas y herramientas definidas actualmente para procesamiento paralelo son muy generales por lo que las soluciones que se crean requieren de mucho trabajo de implementación. La plataforma Génesis como una herramienta especializada en el procesamiento paralelo aplicado al tratamiento de grandes cantidades de datos necesita mecanismos de persistencia de la información, herramientas acopladas que le permitan acceso concurrente a Bases de Datos, rendimiento y alta disponibilidad del servicio de Bases de Datos para lograr los niveles de acceso que se requieren. Este trabajo tiene como objetivo diseñar e implementar una capa de acceso concurrente a datos distribuidos soportados con servicio de alta disponibilidad para Génesis.

Índice general

Índice de figuras

Índice de cuadros

Introducción

La gestión del conocimiento, los avances de las Tecnologías de la Informática y las Comunicaciones y su expansión casi instantánea a través de la Internet, propician las condiciones necesarias para un desarrollo acelerado de las investigaciones. Los procesos de negocio involucran cada día mayores volúmenes de información y la complejidad de sus relaciones evoluciona de manera creciente por lo que las empresas e instituciones científicas, apuestan por mejorar o crear nuevas alternativas eficientes y menos costosas para la modelación de los mismos, unas por no quedarse al margen de la competencia, otras por aumentar su nivel de desarrollo asumiendo proyectos de gran envergadura que necesitan de soluciones científicas elegantes. En muchos casos la solución existe, pero los métodos son ineficaces. Existen problemas cuya modelación computacional introduce un costo de tiempo considerable, otros que simplemente su solución no puede ser asumida debido a la tecnología que exige, unas veces muy costosa y en otras inexistente. De ahí que muchas universidades proyecten sus investigaciones hacia el procesamiento paralelo de datos. La computación paralela se refiere al uso de varios procesadores trabajando juntos para hacer una tarea en común, donde cada procesador trabaja en una porción de un problema que puede ser dividido en pequeñas partes, y entre ellos pueden intercambiar datos a través de la memoria o por una red de interconexión. En la actualidad una de las mayores aplicaciones de la computación paralela es la capacidad de procesar grandes volúmenes de datos en tiempos moderados. Esto genera gran impacto sobre una amplia variedad de áreas, desde simulaciones para ciencia e ingeniería hasta aplicaciones comerciales en minería de datos y procesamiento de transacciones, aplicaciones científicas de bioinformática, análisis de secuencias biológicas, aplicaciones de física computacional como predicción meteorológica, y aplicaciones de análisis para optimizar negocios y decisiones de mercado. Las múltiples arquitecturas definidas actualmente permiten flexibilidad para soluciones específicas de procesamiento paralelo, tienen pocas funcionalidades reutilizables por lo que el desarrollo de aplicaciones en muchos casos se vuelve engorroso. La Plataforma de Análisis y Procesamiento Paralelo de Datos Génesis, actualmente en desarrollo en la Universidad de las Ciencias Informáticas, constituye

una plataforma integral para el desarrollo de aplicaciones de este tipo. Integra un conjunto de soluciones genéricas que con la incorporación de un módulo de cómputo específico se convierte en una solución puntual para un problema determinado de procesamiento paralelo. Génesis está concebida en tres partes fundamentales: un Motor de Procesamiento de Peticiones Paralelas, que constituye el núcleo que planifica y controla todos los procesos de cálculo; un Subsistema de Datos, cuyo objetivo es garantizar la disponibilidad, accesibilidad y persistencia de los datos ofreciendo soporte para el procesamiento de grandes volúmenes de datos en la plataforma y una Interfaz de Administración y Monitorización encargada de la validación y presentación de los datos, la administración y reportes de información del sistema. Génesis esta formada por un conjunto de herramientas, especificaciones y partes de software que pueden reutilizarse a la hora de enfrentar un problema que requiera procesamiento paralelo. Un módulo de cómputo es la documentación e implementación de un modelo matemático que plantea una solución paralelizable a un determinado problema y cuya implementación se basa en las especificaciones que ofrece Génesis. Esta plataforma actualmente no garantiza la disponibilidad, accesibilidad y persistencia de los datos que manipula, necesarios específicamente para el funcionamiento del Motor de Procesamiento de Peticiones Paralelas y el soporte para los módulos de cómputo, principalmente aquellos que impliquen manejabilidad de grandes cantidades de datos y los procesos de administración y monitoreo del sistema. A partir del análisis de esta **situación problemática** surge la presente investigación que plantea el **problema**: ¿Cómo garantizar la disponibilidad y acceso a los datos en la plataforma Génesis? El **objeto de estudio** son los servicios de accesibilidad y acceso a Bases de Datos distribuidas. El **campo de acción** es la disponibilidad y acceso a Bases de Datos distribuidas en la plataforma Génesis.

Objetivo general

Diseñar e implementar una capa de acceso concurrente a datos soportados con servicio de alta disponibilidad.

Objetivos específicos del trabajo.

1. Diseñar e implementar un mecanismo de persistencia para la plataforma Génesis.
2. Diseñar e implementar la capa de acceso concurrente a datos del Motor de Procesamiento de Peticiones Paralelas.
3. Garantizar alta disponibilidad del servicio de acceso a datos en la plataforma.

Para guiar la investigación propuesta se plantea la siguiente **hipótesis**:
Con la implementación de una capa de acceso concurrente a datos distribuidos que cuenten con servicio

de alta disponibilidad se garantizará la alta disponibilidad y accesibilidad a los datos en la plataforma Génesis.

Tareas a desarrollar:

1. Fundamentación teórico-metodológica del problema.

    a) Revisión de la bibliografía relacionada con el tema.

    b) Estudio de los principios básicos de programación en entornos multihilos y acceso concurrente a Bases de Datos.

    c) Análisis de las técnicas y tecnologías utilizadas para la replicación de datos actuales y estudio de las tendencias alrededor de este tema.

    d) Estudio de las tecnologías actuales para garantizar servicios de alta disponibilidad.

2. Elaboración de la propuesta.

    a) Análisis y selección de las herramientas, tecnologías y arquitecturas a utilizar para la elaboración de la propuesta de solución.

    b) Diseño e implementación de un Motor de Persistencia de datos acoplado a la plataforma Génesis.

    c) Diseño e implementación de la capa de acceso concurrente a datos distribuidos.

    d) Diseño del modelo de datos correspondiente de la plataforma que contribuya a la rapidez de las operaciones de manejo de los datos almacenados.

    e) Aplicación de técnicas de replicación y alta disponibilidad de datos con balance de carga al clúster de Bases de Datos.

    f) Instalación y configuración de herramientas para garantizar la disponibilidad de los servicios del clúster de Bases de Datos.

3. Redacción del informe de investigación.

El presente trabajo consta de Introducción, 4 capítulos, 25 epígrafes, 18 subepígrafes, Conclusiones, Recomendaciones, Referencias Bibliográficas y un Glosario de Términos.

En el **Capítulo 1 Fundamentación Teórica** se hace un resumen acerca de las principales tecnologías y tendencias actuales del acceso a Bases de Datos, los procesos de réplica, y los servicios de alta disponibilidad. También se describen las herramientas utilizadas en la propuesta de solución.

En el **Capítulo 2 Motor de Persistencia de Datos para la plataforma Génesis** se presenta el diseño de clases del Motor de Persistencia y su descripción. Se destacan sus principales características, ventajas y desventajas asociadas al diseño y se enuncian las especificaciones para la incorporación de nuevas librerías de conexión.

En el **Capítulo 3 Capa de Acceso a Datos de la plataforma Génesis** se describe la capa de acceso a datos de la plataforma Génesis de las aplicaciones Motor de Procesamiento de Peticiones Paralelas en modo Servidor y en modo Cliente, se muestran los requerimientos en los que se basa el diseño de clases persistentes y de la base de datos. Se describe parte del diseño de los componentes de estas aplicaciones que acceden a la base de datos y se documentan las especificaciones para la adición de módulos de cómputo.

En el **Capítulo 4 Disponibilidad de Datos** se describe el Modelo de Despliegue de la plataforma, y la alternativa propuesta para garantizar la disponibilidad y el rendimiento del SGBD validada por un Caso de Pruebas de Estrés.

Capítulo 1

Fundamentación Teórica

1.1. Introducción

En el presente capítulo se hace un resumen de las principales tendencias que existen en la actualidad para el acceso a datos almacenados en múltiples plataformas, la replicación de datos y las tecnologías existentes para garantizar alta disponibilidad de servicios. Al final del capítulo se analizan las herramientas y técnicas utilizadas para la elaboración de la propuesta de solución.

Los Datos.

Es evidente que los datos son un activo clave, tanto para un usuario de empresa, corporativo como doméstico. Los clientes quieren que sus datos estén siempre disponibles; deben estar seguros; el acceso debe ser de alto rendimiento; sus aplicaciones deben ser escalables y de fácil soporte.

Las aplicaciones de software que gestionan datos almacenados tratan de solucionar al máximo cuatro requerimientos fundamentales con el objetivo de satisfacer las necesidades de sus clientes:

1. La persistencia de los datos.
2. El acceso eficiente a los datos.
3. Rapidez de ejecución de operaciones sobre los datos.
4. Alta disponibilidad del servicio de acceso a los datos.

1.2. Persistencia de los datos.

La persistencia de la información es una de las partes críticas de las aplicaciones de software. "*La persistencia de los datos es la habilidad que tiene un objeto de sobrevivir al ciclo de vida del proceso en el cual reside. Los objetos que mueren al final de un proceso se llaman transitorios.*"[1]

La persistencia es completamente ortogonal al tipo de datos. En otras palabras, la persistencia es una propiedad de los objetos, no de sus clases. [6] La ortogonalidad se refiere al derecho que tienen todos los objetos de persistir independientemente de su tipo de datos, aunque al persistir un objeto, también lo hace su tipo, no se debe almacenar un valor de un tipo determinado y leerlo como otro.

Grady Booch define a la persistencia de la siguiente manera: "*La persistencia es la propiedad de un objeto por la que su existencia trasciende el tiempo (es decir, el objeto continúa existiendo después de que su creador deja de existir) y/o el espacio (es decir, la posición del objeto varía con respecto al espacio de direcciones en el que fue creado)* "[7]

Aunque algunos relacionen la persistencia de los objetos directamente al uso de Bases de Datos, existen muchas formas que resultan útiles en dependencia de la aplicación que se crea. En las aplicaciones orientadas a objetos, tecnología bastante popular en el diseño de software actual, la persistencia se logra principalmente a través de la serialización de los objetos, o el almacenamiento en Bases de Datos. [8]

Cuando son necesarias ciertas características como la independencia de los datos y los programas que los manipulan, la eliminación de la redundancia en el almacenamiento, la sincronización y la integridad de los datos almacenados, mecanismos de seguridad y recuperación y facilidad de manejo de la información transparente a las complejidades de su almacenamiento, la utilización de las Bases de Datos para garantizar la persistencia es la solución mas óptima.

Una base de datos es una "*colección o depósito de datos integrados, almacenados en soporte secundario (no volátil) y con redundancia controlada. Los datos, que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de ellos, y su definición (estructura de la base de datos) única y almacenada junto con los datos, se ha de apoyar en un modelo de datos, el cual ha de permitir captar las interrelaciones y restricciones existentes en el mundo real. Los procedimientos de actualización y recuperación, comunes y bien determinados, facilitarán la seguridad del conjunto de los datos.*"[3]

Es importante diferenciar el término de Bases de Datos a los Sistemas de Gestión de Base de Datos (SGBD). "*Un SGBD es la herramienta que permite interactuar los datos con los usuarios de los datos,*

de forma que se garanticen todas las propiedades definidas en una base de datos."[5] Dichas propiedades se refieren, entre otras, a la integridad, confidencialidad y seguridad.

La tecnología de Bases de Datos se ha consolidado en la actualidad como una solución eficiente para garantizar la persistencia de la información.

1.3. Acceso a Bases de Datos

El acceso a los objetos persistidos es un elemento muy relacionado a la persistencia. No solo se trata de la mejor solución para guardar la información, lograr métodos de acceso tan óptimos como sea posible, es una de las prioridades de los desarrolladores.

El acceso a datos es la "parte" del software que media entre el resto de las "partes" y la base de datos. Un ejemplo muy ilustrativo son las aplicaciones con arquitectura en capas donde una capa es un "*conjunto de subsistemas que comparten el mismo grado de generalidad y de volatilidad en las interfaces: las capas inferiores son de aplicación general a varias aplicaciones y deben poseer interfaces más estables, mientras que las capas más altas son más dependientes de la aplicación y pueden tener interfaces menos estables.*"[9]

La capa de acceso a datos es la que ofrece un conjunto de interfaces que permiten la abstracción al programador de las especificidades del proveedor de datos, siendo una de las capas inferiores mencionadas anteriormente. Si ocurren cambios en la estructura de almacenamiento, o sea la base de datos, las modificaciones afectarán solo esta capa de la aplicación y el resto de las capas del programa no sufrirán modificaciones.

Los métodos de implementación de la capa de acceso a datos son tan variables como las aplicaciones que se desarrollan y dependen específicamente de los problemas que surgen a partir de las características de las mismas. En el caso de las aplicaciones multihilos con implementación orientada a objetos se identifican dos problemas fundamentales en este medio:

1. La concurrencia en el acceso a los datos almacenados.

2. La interacción entre el modelo relacional de la base de datos y el modelo orientado a objetos de los programas.

1.3.1. Concurrencia en el acceso a datos

La concurrencia es el concurso simultáneo de varias circunstancias [4], o sea, la simultaneidad de hechos. Para aplicaciones multihilos que interactúan con Bases de Datos multiusuario, la concurrencia se analiza en dos escenarios: el acceso concurrente a través de múltiples conexiones de usuarios al mismo conjunto de datos y la manipulación concurrente de una misma conexión por múltiples hilos de ejecución de la aplicación.

Acceso concurrente de múltiples usuarios.

Los SGBD multiusuario garantizan la concurrencia en la ejecución de varias transacciones casi instantáneas sobre el mismo conjunto de datos. Estos SGBD se caracterizan por las propiedades ACID de las transacciones que manipulan:

1. **Atomicidad (Atomicity):** Todas las operaciones de una transacción son ejecutadas por completo, o no se ejecuta ninguna de ellas.

2. **Consistencia (Consistency):** Una transacción T transforma un estado consistente de la base de datos en otro estado consistente, aunque T no tiene por qué preservar la consistencia en todos los puntos intermedios de su ejecución.

3. **Aislamiento (Isolation):** Una transacción está aislada del resto de transacciones. Aunque existan muchas transacciones ejecutándose a la vez, cualquier modificación de datos que realice T está oculta para el resto de transacciones hasta que T sea confirmada.

4. **Durabilidad (Durability):** Una vez que se confirma una transacción, sus actualizaciones sobreviven cualquier fallo del sistema. Las modificaciones ya no se pierden, aunque el sistema falle justo después de realizar dicha confirmación.

El Subsistema de Recuperación del SGBD es el encargado de conseguir el cumplimiento de las propiedades de atomicidad y durabilidad de las transacciones. La conservación de la consistencia es una propiedad cuyo cumplimiento han de asegurar, por un lado los programadores de base de datos, y por otro el Subsistema de Integridad del SGBD. El Subsistema de Control de Concurrencia es el encargado de conseguir el aislamiento de las transacciones. Para este caso los programadores tienen que garantizar únicamente que sus transacciones no dejen estados inconsistentes en la base de datos evitando la redundancia de la información y logrando un buen diseño del modelo de datos.

Manipulación concurrente de una conexión por múltiples hilos de ejecución.

Los mecanismos de sincronización son una parte fundamental en el diseño de las aplicaciones multihilos, sobre todo cuando un recurso determinado puede ser utilizado de manera concurrente. Para el acceso a datos en este escenario, se introduce el problema de que una conexión pueda ser manipulada por varios hilos de ejecución y no se apliquen mecanismos de sincronización que impidan la colisión de las operaciones. Existen librerías para acceso a Bases de Datos desde lenguajes de programación que no son "thread-safety", otras que lo son pero imponen restricciones en su utilización.

Para garantizar la sincronización en el acceso a la base de datos de aplicaciones multihilos se proponen varias soluciones:

1. **Asignar una conexión nueva para cada hilo de ejecución.** *(Figura 1.a)*

   Todas las transacciones ejecutadas por un hilo de ejecución utilizarán una conexión propia. Este método es fácil de elaborar, cada hilo utiliza una conexión diferente por lo que no se producen colisiones en la ejecución de las transacciones. El patrón de diseño Singleton [10] puede ser utilizado para garantizar que la conexión sea única en cada implementación de los hilos. En este caso es medible la cantidad de conexiones a priori que se realizarán en una aplicación si se conoce la cantidad de hilos de ejecución que se levantan y acceden a la base de datos. Pero la existencia de conexiones abiertas que no se estén utilizando en intervalos de tiempo largos no es eficiente por lo que se recomienda solo si la cantidad de hilos que necesitan de una conexión nueva es mínima y se ejecutan operaciones sobre la base de datos constantemente.

2. **Compartir conexiones entre hilos de ejecución.** *(Figura 1.b)*

   Cuando las operaciones de acceso que se realizan en un mismo hilo de ejecución distan unas de otras, puede optimizarse el uso de las conexiones. Para ejecutar una operación, se hace la conexión, se realizan las consultas necesarias y luego se cierra y puede ser aprovechada quizás por otro hilo que la necesite, así solo se utiliza en el momento que es requerida. Cuando se necesite de nuevas operaciones entonces se realiza el mismo procedimiento, siempre tratando de ejecutar el mayor número de operaciones posibles para reducir el costo de la conexión y la desconexión por cada consulta realizada.

   Podemos realizar pruebas de estrés al SGBD para conocer el límite de conexiones que puede manejar manteniendo la capacidad de respuesta requerida. Así se crean y se administran esas conexiones y podremos lograr mejor rendimiento. Es responsabilidad del programador cuando se

alcanza un límite de conexiones determinado, acumular las peticiones nuevas que llegan hasta que puedan ser atendidas y no se pierdan los datos. Las librerías para conexión en su mayoría tienen funciones que permiten asignar múltiples conexiones de manera no-bloqueante[1]. Si el SGBD está distribuido y puede soportar grandes cargas de operaciones, no se recomienda limitar el número de conexiones sino explotar las capacidades del SGBD.

3. **Aplicar mecanismos de bloqueo de recursos.** *(Figura 1.c)*

   Puede garantizarse la sincronización de los hilos de ejecución que acceden a la base de datos mediante el uso de semáforos. Se crea una conexión única con el patrón Singleton, cuando va a ser utilizada por un hilo de ejecución se bloquea un semáforo, se realizan las operaciones y se desbloquea el semáforo. Si llegaron nuevas peticiones de uso de la conexión todos los semáforos bloqueados se ponen en cola de espera y a medida que se van liberando, el próximo en la cola continúa su procesamiento. Este método solo es aplicable para librerías que impongan restricciones solo sobre la conexión y no es eficiente para sistemas paralelos porque las operaciones sobre la base de datos se realizan secuencialmente, y no se aprovecha al máximo el rendimiento del SGBD.

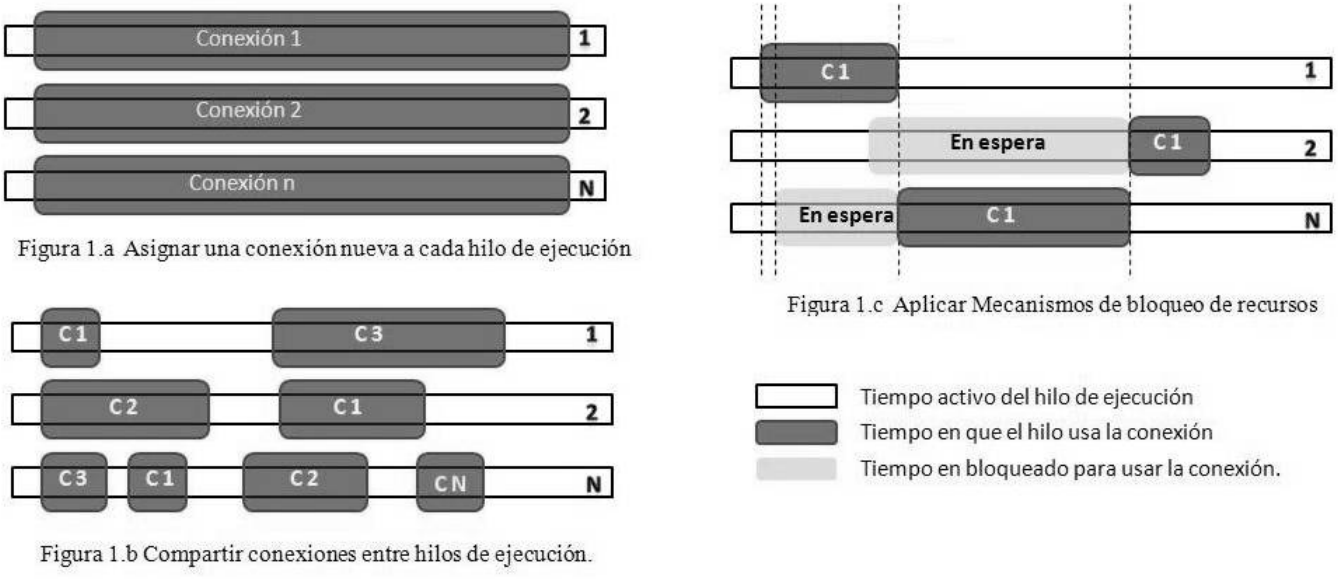

Figura 1.1: Sincronización entre hilos de ejecución para acceso a Bases de Datos.

[1]Ejemplo son las funciones "PQconnectStart" y "PQconnectPoll" de la librería libpq para conexión a PostgreSQL desde C++.

La concurrencia en el acceso a datos es un factor que influye decisivamente en el diseño e implementación de este tipo de sistemas.

1.3.2. Interacción entre el modelo relacional de la base de datos y el modelo orientado a objetos de las aplicaciones que desarrollamos.

Modelo Orientado a Objetos

Los sistemas basados en modelos orientados a objeto fueron inspirados a partir del paradigma de programación orientada a objetos. El primer lenguaje de programación orientado a objetos fue Simula 67 oficialmente presentado por Ole Johan Dahl y Kristen Nygaard en la Conferencia de Lenguajes de Simulación en Lysebu en Mayo de 1967. [11, 12] Sin embargo, el primer lenguaje que popularizó la aproximación a objetos fue Smalltalk (1976). Posteriormente aparecieron lenguajes orientados a objetos más avanzados como C++ en 1980 diseñado por Bjarne Stroustrup [17] y Java desarrollado por la Sun Microsystems a principio de los años 90.[13]

Algunos conceptos asociados al Modelo Orientado a Objetos son:

1. **Clase:** Define el comportamiento y la estructura de un tipo de objetos determinado. Es una plantilla para crear muchos objetos independientes con las mismas características.
2. **Objeto**: Unidad creada a partir de una clase que combina sus propiedades y funciones. Cada objeto recuerda sus propios valores y representa una instancia de una clase.
3. **Atributo**: Propiedad de los objetos que pueden adquirir valores de un dominio.
4. **Funciones**: Acciones, métodos que puede realizar un objeto.
5. **Identidad del objeto:** Permite identificar a un objeto del resto aún cuando pertenecen a la misma clase y tengan igual estado (valores de sus atributos).
6. **Encapsulamiento:** Ocultamiento de los atributos y las acciones de los objetos. Oculta los detalles de su implementación.
7. **Abstracción:** Denota las características esenciales que distinguen a un objeto de otros tipos de objetos, definiendo precisas fronteras conceptuales, relativas al observador.

Modelo relacional

La popularidad del modelo relacional se debe primariamente a su simplicidad. Aparece en 1970 presentado por Edgar F. Codd en el artículo "A relational model for large shared data banks" publicado en "Communications of the ACM"[19]. Representó un verdadero hito en el surgimiento de los Sistemas de Gestión de Bases de Datos (SGBD). Sobre el modelo relacional se han definido los estándares ANSI e ISO del lenguaje de definición y manipulación de Bases de Datos relacionales SQL (acrónimo de Structured Query Language).[14] A partir de mitad de los años 80 hasta la actualidad el modelo relacional es utilizado por la mayoría de los SGBD y se pueden destacar algunos muy buenos como: PostgreSQL, MySQL (libres) y Oracle, Microsoft SQL Server (no libres).

Algunos conceptos asociados al modelo relacional son [14]:

1. **Relación:** Representa tanto instancias de una entidad del universo real como interrelaciones entre entidades de distinto tipo. Su representación informal es una tabla. La relación es el elemento fundamental del modelo relacional.

2. **Dominio:** Conjunto válido de valores de referencia para definir propiedades o atributos. Un dominio es un conjunto nominado y homogéneo de valores que pueden adquirir los atributos de las relaciones.

3. **Atributo:** Representa una propiedad de una relación y tiene dependencia existencial de la misma. Toma valores de un dominio. Su representación informal es una columna.

4. **Tupla:** Ocurrencia o instancia dentro de una relación. Permite referenciar una instancia de una entidad en el universo o la interrelación específica o concreta entre instancias de entidades. Su representación informal es una fila. Una relación tiene un conjunto de tuplas.

5. **Clave primaria:** Identificador único para cada tupla que garantiza que nunca existen dos tuplas con los mismos valores de sus atributos.

1.3.3. Un problema, varias soluciones.

Las diferencias entre ambos modelos, conocida como Desajuste por Impedancia[18], es notable y a medida que el modelo de objetos aumenta las incongruencias son cada vez mayores. En este caso se referencian un conjunto de soluciones que adoptan los arquitectos en el diseño de aplicaciones orientadas a objetos.

1. **Entorno puramente relacional:** No existe un modelo de clases persistentes orientado a objetos. La lógica de la aplicación interactúa directamente con la base de datos o reside generalmente en los procedimientos almacenados en el SGBD. Los programadores están familiarizados con el lenguaje SQL por lo que podría pensarse que no hay problemas, sin embargo las complejidades en el código aumentan considerablemente, las aplicaciones son poco portables y difíciles de mantener.

2. **Entorno puramente Orientado a Objetos:** Sustituir el SGBD relacional u objeto relacional por un SGBD Orientado a Objetos (SGBDOO) nos permitiría un mayor acercamiento del modelo de datos al diseño orientado a objetos de nuestras aplicaciones. Las Bases de Datos orientadas a objetos (BDOO) son aquellas cuyo modelo de datos está orientado a objetos y almacenan y recuperan objetos de los que se almacena su estado y comportamiento. [15]Las clases utilizadas en la aplicación son las mismas clases que serán utilizadas en una BDOO; de tal manera, que no es necesaria una transformación del modelo de objetos para ser utilizado por un SGBDOO.

   Aunque los SGBDOO proporcionan soluciones apropiadas para un gran número de aplicaciones no tienen el nivel de desarrollo de los SGBD relacionales y los SGBD objeto-relacionales que se han consolidado como soluciones factibles en el desarrollo de software en la actualidad.

   El modelo relacional tiene una sólida base teórica y los productos relacionales disponen de muchas herramientas de soporte que sirven tanto para desarrolladores como para usuarios finales. El modelo estándar ODMG, con un lenguaje de definición de objetos ODL y un lenguaje de consulta a objetos OQL es bastante eficiente pero aún no tiene fundamento matemático teórico que lo sustente.

   Otro aspecto es la optimización de consultas que requiere una compresión de la implementación de los objetos, para mejorar los tiempos de acceso a la base de datos, sin embargo, se compromete el concepto de encapsulamiento.

   Respecto a las herramientas existentes, se destaca BD4Objects como una de las soluciones más experimentadas, BDOO libre bajo la licencia GPL, desarrollada para Java y tecnología .NET. [16]

3. **Mapeo[2] Objeto-Relacional:** (ORM, acrónimo de Object-Relational Mapping).

[2]El término se refiere a la traducción del término ***Mapping*** en Inglés que es utilizado en toda la bibliografía consultada para referirse al proceso de hacer corresponder el modelo relacional y el modelo orientado a objetos. En el documento lo utilizaremos con la misma intención para ganar en comprensión.

Los ORM permiten la coexistencia de el modelo relacional de la base de datos y el diseño orientado a objetos de las aplicaciones en nuestros entornos de software y la comunicación entre ellos, garantizando cierta abstracción del programador sobre el modelo relacional pudiendo manipular la persistencia de sus objetos con una mínima interacción con el mismo. En su mayoría están compuestos por librerías, clases y formas descriptivas que permiten el mapeo de las clases. Soportan al menos el mapeo a un tipo de Bases de Datos, son utilizados únicamente por las aplicaciones implementadas con el mismo lenguaje de programación y, por supuesto, mapean Bases de Datos relacionales u objeto-relacionales únicamente.

Un buen ORM debe permitir[8]:

a) Mapear las clases del modelo Orientado a objetos a las tablas del modelo relacional y las propiedades a columnas.

b) Persistir objetos a través de un método Orm.Insertar (objeto) encargándose de generar el código SQL correspondiente.

c) Recuperar objetos persistidos a través de un método objeto = Orm.Cargar (objeto.clase, clave_primaria).

d) Recuperar una lista de objetos a partir de un lenguaje de consulta especial a través de un método: ListaObjetos = Orm.Buscar (Objeto FROM MiObjeto WHERE Objeto.Propiedad=5), o algo más complejo ListaObjetos = Orm.Buscar (Objeto ORM FROM MiObjeto WHERE Objeto.Relacion1.Relacion2.Propiedad2=5), y el ORM transformará a través de varias consultas entre tablas.

Los ORM pueden clasificarse en dependencia del tipo de mapeo que realizan:

a) **Mapeo de Objetos Ligero:** Las entidades se presentan como entidades que se mapean manualmente a las tablas en la base de datos. Las llamadas al RDBMS se separan de la lógica de negocio a través de algún patrón de diseño como el DAO, acrónimo de Data Access Object. Las consultas complejas se elaboran en lenguaje SQL y se ejecutan directamente. Esta estrategia es muy común y tiene éxito en aplicaciones con entidades sencillas.

b) **Mapeo de Objetos Medio:** Es diseñado completamente a través de los objetos que persisten. Cubre las funcionalidades del anterior y además la persistencia de colecciones de objetos. Las sentencias SQL se generan en tiempo de ejecución. Adecuado para aplicaciones de complejidad media y que necesiten de cierta portabilidad a varios SGBD.

c) **Mapeo de Objetos Completo:** Este método es una evolución de los anteriores, ofrece soporte para relaciones complejas como la herencia y ofrece un lenguaje completo de consultas orientado a objetos. Cubre las relaciones de carga perezosa (lazy loading) para cargar los objetos persistidos, sin el grafo de sus relaciones, solo hasta que sean solicitados y carga activa (eager loading) que extrae un objeto con el grafo de sus relaciones completo. Este nivel de funcionalidad es ciertamente difícil de conseguir y puede demorar bastante tiempo de implementación y prueba.

   Una solución ORM característica de ese método es Hibernate[2], catalogado como una de las mejores soluciones de este tipo para Java.

Aunque los ORM completos sean una solución bastante atractiva, porque han aportado muy buenos resultados para muchas aplicaciones tienen el problema de que los lenguajes especiales de consultas entre objetos introducen lentitud en la ejecución de las operaciones en comparación con SQL, tienden a introducir demasiada arquitectura en casos donde la solución es mucho más simple.[20, 21]

Se realizó énfasis en estos el problema de la concurrencia y la interacción entre los modelos relacional y orientados a objetos que son los más influyentes en la programación multihilo orientada a objetos, aunque existen otros que no son tan específicos de esta rama pero que influyen como el volumen de recuperación de datos, la seguridad y la portabilidad. Elegir una solución específica para implementar nuestra capa de acceso a datos depende de los requerimientos del software que diseñamos y de la disponibilidad y características de herramientas que podemos utilizar. Sin embargo muchos prefieren desarrollar sus propios métodos adaptados a las necesidades del software que desarrollan, logrando mayores índices de eficiencia.

1.3.4. Algunas soluciones para acceso a datos con C++.

En esta sección se presentan algunas soluciones para acceso a datos desde C++ con las principales características que le distinguen.

Debea: Debea es una colección de interfaces que permite mapear objetos a relaciones a aplicaciones desarrolladas en C++. Para crear los objetos es necesario crear el código SQL correspondiente. No elimina completamente el código SQL para la realización de consultas complejas. Tiene soporte para Bases de Datos: SQLite3, PostgreSQL, todas las Bases de Datos que pueden ser accedidas usando

ODBC o iODBC. Tiene una arquitectura flexible para agregar soporte a nuevos SGBD. No es "thread-safety". Es multiplataforma. [22]

SOCI: SOCI (Simple Oracle Call Interface) es una librería para acceso a Bases de Datos con C++ que simula la inserción de consultas SQL en la sintaxis del código C++ sin violar las restricciones del Estándar C++. Actualmente en su versión 3.0 ofrece soporte para los servidores de Bases de Datos PostgreSQL, Oracle y MySQL. Es fácil de utilizar, pero no es robusta en cuanto al soporte de tipos de datos no propios de C++, por lo que requiere de mucha implementación adicional para lograr un buen diseño del acceso. SOCI no es "thread-safety" y sus instancias no pueden ser accedidas concurrentemente. La solución que ofrece para estos casos son funciones para administrar múltiples conexiones a la vez. [23]

DTL: DTL (Database Template Library) es una librería para el acceso a Bases de Datos que proporciona los datos en estructuras que pueden ser tratadas igual a las estructuras STL , acrónimo de Standard Template Library de C++. Los cambios que se realicen en dichas estructuras, serán actualizados inmediatamente en la base de datos. Se compila con la librería STL, por lo que se pueden reutilizar, los algoritmos para almacenamiento, búsqueda y manipulación de la información. Ofrece soporte para Bases de Datos con Oracle, Microsoft SQL Server 2000, Access y MySQL. Sus desarrolladores no ofrecen seguridad sobre el soporte para PostgreSQL así como no garantizan que la librería sea "thread-safety". [24] No solo impone restricciones sobre la conexión sino que las estructuras recuperadas no pueden ser manipuladas por varios hilos de ejecución.

LiteSQL: Es un ORM para implementar la persistencia de datos en C++. Útil para servidores de Bases de Datos LiteSQl, MySQL y PostgreSQL. Garantiza la persistencia de los objetos y sus relaciones. Minimiza, aunque no suprime la necesidad de ejecutar consultas SQL por los programadores. Necesita de creación de XML para la definición de objetos y no es "thread-safety"[25]

1.4. Distribución de los datos.

Se ha demostrado que la persistencia y los métodos de acceso son factores estrechamente relacionados que influyen en el rendimiento de la manipulación de los datos de las aplicaciones, pero ¿son los únicos? ¿qué sucede cuando la cantidad de usuarios o de operaciones que se realizan sobre los datos almacenados es tal que buenos métodos de acceso o de almacenamiento no son capaces de garantizar el rendimiento deseado?

En determinados casos la capacidad de respuesta y la disponibilidad de los sistemas aumenta con-

siderablemente cuando los datos que manipulan se encuentran distribuidos.

1.4.1. Bases de Datos distribuidas.

Las Bases de Datos distribuidas son consideradas como Bases de Datos implementadas en la red. Los componentes se distribuyen sobre varios nodos (estaciones) de la red. Dependiendo de la actualización específica y del tráfico de recuperación, distribuir una base de datos puede aumentar significativamente el rendimiento general.

Las Bases de Datos distribuidas surgieron como una solución eficiente para las aplicaciones que necesitaban alta disponibilidad en el servicio del acceso a datos. Su objetivo primordial es mejorar la eficiencia de su desempeño físicamente distribuyendo los datos de acuerdo a los requerimientos de uso y las capacidades computacionales que disponen los diferentes usuarios, pero manteniendo la visión del sistema como un solo componente.[33]

Las principales alternativas existentes para la distribución de los datos son la fragmentación y la replicación.

Fragmentación:

En la fragmentación el modelo de datos se encuentra distribuido, es decir se parten las relaciones en pequeñas relaciones o fragmentos y son almacenados en diferentes locaciones. Estos fragmentos contienen suficiente información como para permitir la reconstrucción de la relación original. Este modelo puede realizarse de tres formas diferentes: partición vertical, partición horizontal y mixta; referidos a la fragmentación por campos de las tablas, por tuplas o una combinación de ambos respectivamente. La ventaja significativa es que las consultas pueden ser particionadas también por lo que su procesamiento es paralelo y más eficiente, sin embargo esta técnica sacrifica el rendimiento en a escenarios de operaciones y la disponibilidad ante el fallo de una de las Bases de Datos acopladas. Es muy complejo.

Réplica de datos:

En este esquema cada nodo contiene exactamente una copia del modelo de datos completo. Cuando se realizan operaciones de escritura en uno de los servidores, se requiere de rápida actualización de los datos en todos los nodos, lo que asocia un costo en el rendimiento respecto al método anterior. Sin embargo el rendimiento respecto al modelo de no distribución de datos es considerablemente mayor y se garantiza la disponibilidad y fiabilidad de los datos.

La replicación posee las siguientes características:

1. **Rendimiento:** Provee un rápido acceso local a datos compartidos ya que balancea la actividad sobre múltiples nodos. Algunos usuarios pueden acceder a un servidor mientras que otros acceden a diferentes servidores, reduciendo la carga de todos los servidores.

2. **Tolerancia a fallos:** Ofrece opciones alternativas de acceso a los datos. Si un nodo no se encuentra disponible el servicio no se afecta los usuarios pueden continuar realizando sus operaciones en las locaciones restantes. Cuando el nodo fallido se incorpora es actualizado y continúa prestando servicio.

3. **Escalabilidad:** La escalabilidad se logra hasta cierto punto, en dependencia de los entornos de configuración. A medida que se incrementa el número de servidores la prestación de servicios es mayor porque la carga de procesamiento individual disminuye, pero el tráfico en la red aumenta lo que significa disminución del rendimiento. Es necesario desarrollar pruebas que nos permitan seleccionar la mejor configuración.

Un elemento muy importante a tener en cuenta es que cualquier sistema debe asegurar que todas sus réplicas de una relación n sean consistentes; en caso contrario pueden producirse cómputos erróneos. Siempre que se actualice n, se deben actualizar también todas sus réplicas, aunque este proceso pueda llegar a representar en algún momento una sobrecarga de la red durante la actualización.

1.4.2. Entornos de Réplica

Entorno de replicación maestro-esclavo:

Conocida también como "de solo lectura", permite a un solo maestro recibir consultas de lectura/escritura, mientras los esclavos solo pueden aceptar consultas de lectura. Debido a que las operaciones de escritura pueden ser efectuadas en un único nodo, para sistemas que realicen modificaciones constantemente, no se logran buenos índices de rendimiento.

Entorno de replicación multimaestro:

Conocida como "par a par o la réplica de camino de n", permite múltiples nodos, actuando como pares iguales. En este entorno cada nodo es maestro, y se comunica con otros nodos maestros. [34] Cada nodo permite operaciones de lectura y escritura, cuando se modifica cualquiera de los nodos el

resto es actualizado, por lo que se elimina la sobrecarga de un único nodo, como ocurre en el entorno anterior. La replicación multimaestro es una alternativa eficaz para aquellos sistemas que realizan gran cantidad de operaciones de lectura/escritura sobre sus datos.

1.4.3. Tipos de Replicación

Los tipos de replicación están relacionados con el momento en que se realiza la actualización de los datos cuando se efectúa una transacción de escritura en la base de datos .

Replicación Asíncrona:

En el caso de la replicación asíncrona, las actualizaciones a una réplica son propagadas hacia los demás en algún momento posterior, no dentro de la misma transacción, por lo tanto la replicación asíncrona presenta un retardo de tiempo o latencia, durante el cuál es posible que las réplicas no sean idénticas (y por lo tanto el término réplica ya no es muy adecuado, debido a que ya no estamos hablando de copias exactas).[35]

Este tipo de replicación se muestra en varias formas:

1. **De Sitio Primario:** Una copia de la relación es designada como copia maestra o primaria. Las réplicas o los fragmentos de la relación completa pueden ser creados en otros nodos; estas serían copias secundarias, y a diferencia de la copia primaria; pueden no ser actualizadas.

2. **Par a Par:** Más de una copia (aunque no todas) puede ser marcada con permisos de actualización, esta es una copia maestra. Además para propagar los cambios, una resolución de conflicto puede ser usada para lidiar con el hecho de hacer el cambio en los diferentes sitios. Esta es la más utilizada.

Replicación Síncrona

En el caso de la replicación síncrona, si se actualiza una réplica dada, todas las demás réplicas del mismo fragmento de datos también se actualizan dentro de la misma transacción; lo que implica que (desde un punto de vista lógico) solo existe una versión de los datos. La mayoría de los productos implementan la replicación síncrona por medio de procedimientos disparados (posiblemente ocultos y manejados por el sistema). Sin embargo la replicación síncrona tiene la desventaja de que impone una

sobrecarga sobre las transacciones que actualizan cualquier réplica (también puede haber problemas de disponibilidad)[35]

En este caso se aplican dos técnicas de sincronización:

1. **Voting:** Una transacción debe escribir la mayoría de las copias para modificar un objeto y leer al menos suficientes copias para asegurarse que esa copia está presente. Esta técnica no es muy recomendada ya que en la mayoría de los casos leer un objeto requiere leer múltiples copias, y en muchas aplicaciones, los objetos son leídos con más frecuencia que actualizados.

2. **Read any write all:** (Leer de cualquiera y escribir en todos) Para leer un objeto, una transacción puede leer cualquier copia, pero para escribir un objeto, ésta debe escribir todas las copias. Las lecturas son rápidas, especialmente si tenemos una copia local, pero escribir se vuelve muy lento, en relación con la técnica Voting. Esta técnica (read any write all) recomendada cuando las operaciones de lectura son más frecuentes que las de escritura es la más usada a la hora de implementar la replicación síncrona.

1.4.4. Soluciones de replicación existentes

Existen varias soluciones para garantizar la réplica de los datos almacenados. Algunas están implementadas para SGBD específicos, otras son más generales. Para la replicación en PostgreSQL son aplicables un conjunto de soluciones como Slony-I, PgCluster y PyReplica; que se destacan como las más utilizadas.

Estas aplicaciones se dividen de acuerdo a los entornos de réplica donde pueden ser aplicadas, para el entorno "maestro-esclavo", la más popular es:

Slony-I

Es un sistema de replicación "maestro-esclavos", soporta la réplica en cascada y permite a un esclavo ser a su vez maestro para otro servidor. Incluye los tipos de funcionalidades necesarias para replicar Bases de Datos grandes a un número razonablemente limitado de sistemas esclavos. Si el número de servidores crece más allá de ese, el coste de comunicaciones aumentará , y las ventajas incrementales de tener servidores múltiples fallarán en ese punto.

Además permite indicar qué cambios replicar de un servidor a otro. Slony-I implementa la réplica asíncrona, usando disparadores para determinar las actualizaciones de las tablas, donde un solo "origen" (maestro) se puede replegar a los "suscriptores múltiples" (esclavos) incluyendo suscriptores conectados en cascada.[27]

Slony I realiza una réplica de espejos, exactamente igual al origen de datos, no es posible actualizar los datos a medida que se produce algún cambio en ellos. Puede ser útil en determinadas situaciones, pero no da una solución al problema planteado inicialmente.

En entornos "maestro-esclavo" y "multimaestro limitado" se destaca:

PyReplica

PyReplica es un sistema de replicación simple para PostgreSQL desarrollada en Python y multiplataforma. Es asincrónico, sin protocolos especiales ni herramientas externas. En modo Maestro/Esclavo, los esclavos aceptan consultas de solo lectura, se utiliza para copias de respaldo y en Datawarehouses. En modo multimaestro garantiza el servicio para servidores remotos y móviles. Es una tecnología reciente que posee ciertas limitantes en el modo multimaestro fundamentalmente en el tratamiento de los conflictos. [26]

En entornos "multimaestro" la solución más eficiente es:

PgCluster

PgCluster es el sistema de replicación síncrona de composición multimaestro para PostgreSQL basado en consultas. Consiste en tres formas de servidores distintas:

Clúster de base de datos (ClústerBD): Es un nodo máster que contiene una copia exacta de la base de datos. Las operaciones de lectura y escritura se realizan en estos nodos.

Balanceador de carga: Interactúa con la aplicación que accede al SGBD y distribuye las peticiones que recibe a los nodos máster del clúster que tengan menor carga de procesamiento en ese momento.

Servidor de réplica (Replicador): Realizan el proceso de actualización de los datos modificados en los nodos máster que tiene asociados hacia los restantes.

En la *figura 1.1* se presenta una descripción del flujo de eventos entre los servidores anteriores ante la realización de una operación de escritura en el clúster de Base de datos. Es una representación conceptual de las funciones pues un mismo nodo puede ser configurado para ejercer más de un rol.

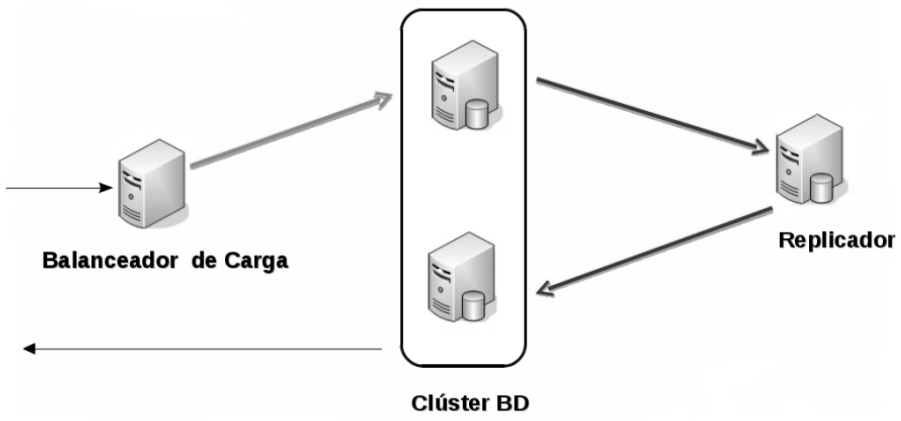

Figura 1.2: Flujo de eventos en un clúster de BD configurado con PgCluster.

Cuando llega una petición de transacción al Balanceador de Carga, este verifica cuál de los nodos del clúster de base de datos tiene menor carga de procesamiento y se la asigna. La transacción se ejecuta y antes de terminar el replicador configurado para ese nodo máster se encarga de actualizar los cambios en los restantes nodos. Una vez completada la transacción y el proceso de copia, los datos de retorno van directamente al nodo que hizo la petición, sin pasar por el Balanceador de Carga.

PgCluster no requiere de hardware especial para el almacenamiento de los datos, utiliza los recursos del servidor en que reside, cuando un máster falla los datos no se pierden porque están replicados en los demás máster del clúster.

PgCluster trae implementadas todas las ventajas del entorno multimaestro y la replicación síncrona, es por eso que para sistemas que necesitan de buenos índices de rendimiento, asociado a servicios de disponibilidad y fiabilidad es una opción recomendable.

1.5. Alta disponibilidad de servicios

La distribución de los datos es un factor importante para optimizar la capacidad de procesamiento de las consultas, pero generalmente se incorporan técnicas que permitan garantizar la disponibilidad del servicio. En la actualidad una de las alternativas para esta problemática son los clúster de alta disponibilidad, que están compuestos por un conjunto de dos o más máquinas y se caracterizan por mantener una serie de servicios compartidos y estar constantemente monitorizándose entre sí, distribuyen una carga de trabajo sobre una gran cantidad de servidores y garantizan que la tarea se realice aún en el

caso de que uno de los nodos falle.

Herramientas para alta disponibilidad

Existen varias herramientas libres que proporcionan este servicio. La selección adecuada de las mismas depende de los objetivos que se persiguen en el desarrollo del software.

Uno de los proyectos líderes en la actualidad en la construcción de herramientas de alta disponibilidad es LinuxHA[32], con el objetivo de proporcionar una alta disponibilidad para Linux que promueve la fiabilidad, disponibilidad y servicio a través de un esfuerzo de desarrollo de la comunidad.

El principal producto que desarrollan y mantienen es HeartBeat, uno de los mejores paquetes de software de la actualidad para Alta Disponibilidad. Se estiman más de treinta mil instalaciones en aplicaciones de misión crítica en todo el mundo desde el año 1999. su página web es visitada casi por veinte mil usuarios en el día y HeartBeat tiene un promedio de tasas descarga de más de cien al día.[32]

¿Por qué utilizar HeartBeat?

Esta tecnología implementa *heartbeats* (latidos del corazón, en su traducción al español). Heartbeat se instala en un servidor y comienza el envío de paquetes constante hacia el servidor original que provee el servicio de aplicación, como se muestra en la *figura 1.3 a)* Si un paquete no llega tras un tiempo establecido, indicaría que el servidor no está disponible, por lo tanto se sabe que interrumpió sus servicios y el servidor secundario se activa asumiendo los servicios del servidor primario, como se muestra en la *figura 1.3.b).*

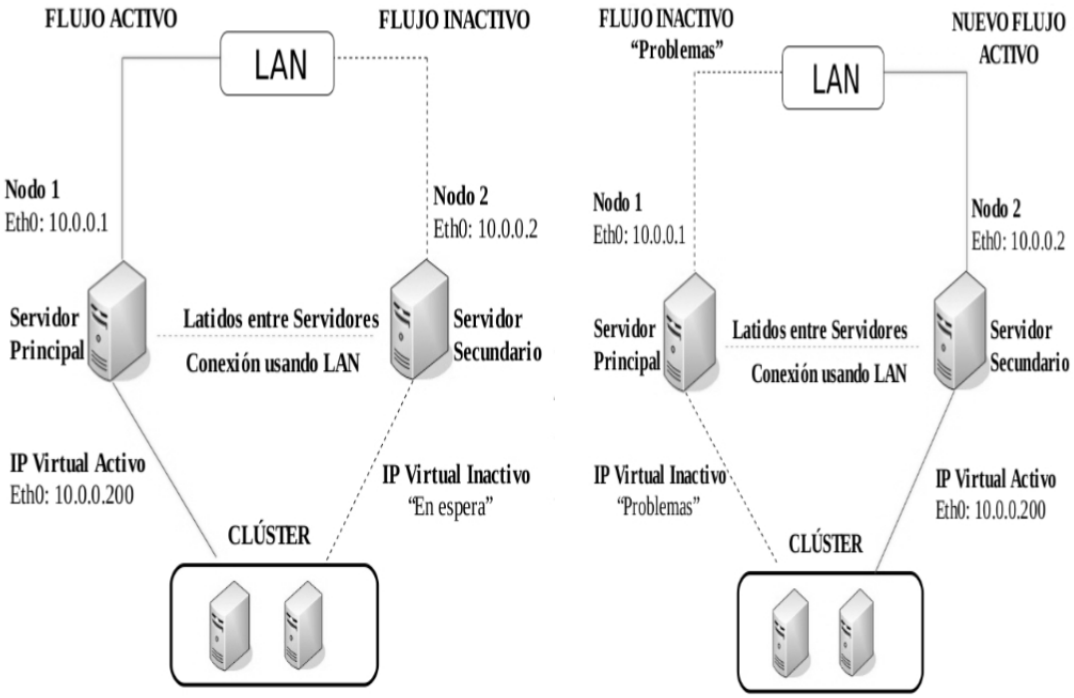

Figura 1.3: Funcionamiento de HeartBeat.

Soporta múltiples direcciones IP y establece una como modelo de recursos, identificando grupos de recursos. Es una aplicación estable que ha sido probada en varias aplicaciones, incluyendo servidores DNS, servidores proxy de cache, servidores web.

Garantiza la seguridad del envío de mensajes permitiendo firmar los paquetes CRC de 32 bits, MD5 y SHA1. De esta manera se evita que un nodo no miembro se enmascare como nodo miembro del clúster y utilice el servicio. Además esta herramienta puede asimilar varias operaciones de mantenimiento de seguridad que necesitan ser efectuadas, como pueden ser cambio de claves y de protocolos de autenticación disponiendo de ficheros para la configuración. [28]

Producto a la arquitectura que presenta y el soporte del software, existen varias herramientas basadas en HeartBeat que implementan alta disponibilidad y otros servicios. Ejemplo son Ldirectord[31] y UltraMonkey [30]

Heartbeat tiene el problema que si no se dispone de una línea dedicada, aunque ésta sea una línea serie, al tener un tráfico que aunque pequeño es constante, suele dar muchas colisiones con otros tráficos que puedan ir por la misma red. A pesar de ello, Heartbeat se ha consolidado como una herramienta altamente estable para garantizar la alta disponibilidad de servicios.

1.6. Tecnologías y herramientas y software para el desarrollo

Sistema Operativo: Distribución Debian GNU/Linux.

Debian es un sistema operativo libre. Utiliza el núcleo Linux , pero la mayor parte de las herramientas básicas vienen del Proyecto GNU; de ahí el nombre Debian GNU/Linux. Ofrece más que un sistema operativo puro, viene con más de 25000 paquetes, programas precompilados distribuidos en un formato que hace más fácil su instalación. Cuenta con varios repositorios en el mundo y específicamente en la Universidad de la Ciencias Informáticas, sus actualizaciones se realizan diariamente. [36]

Lenguaje de programación C++.

El lenguaje de programación C++ más empleados en la actualidad. Se puede decir que C++ es un lenguaje híbrido. Las principales ventajas que presenta el lenguaje C++ son:

1. **Difusión:** Posee un gran número de usuarios y existe una gran cantidad de libros, cursos y páginas web dedicadas a él.
2. **Versatilidad:** C++ es un lenguaje de propósito general.
3. **Portabilidad:** C++ está estandarizado y un mismo código fuente se puede compilar en diversas plataformas.
4. **Eficiencia:** C++ es uno de los lenguajes más rápidos en cuanto ejecución. Muchas aplicaciones están implementadas en C++ .Se destacan Adobe Photoshop e ImageReady, Adobe Acrobat , gran parte del código de Windows XP, Microsoft Office, KDE y muchas otras herramientas libres.[17]
5. **Herramientas:** Existe una gran cantidad de compiladores, depuradores y librerías que permiten el uso de este lenguaje como G++, Borland C++, Microsoft Visual C++.[17]La librería STL (Standard Template Library) brinda estructuras de almacenamiento muy eficientes.

IDE EasyEclipse CDT.

EasyEclipse CDT es un plugin de EasyEclipse para programar en entornos C/C++. EasyEclipse es libre, fácil de descargar e instalar, integrado con un plugin de Subersion (SVN) para el control de versiones y simple de mantener, sin problemas de versiones ni dependencias. Ofrece un editor para

C/C++ con funcionalidades básicas, formato de palabras reservadas del lenguaje y completamiento de código, facilidades de búsqueda, asistente de contenido y generador de ejecutables. [37]Es bastante rápido y consume pocos recursos comparado con sus similares.

Sistema de Gestión de Bases de Datos PostgreSQL.

PostgreSQL es el Sistema de Gestión de Bases de Datos Objeto Relacional de código abierto más popular en la actualidad. Surge como el proyecto Ingres en la Universidad de Berkeley en 1977. En la actualidad continúa un proceso de desarrollo activo a nivel mundial por un equipo de desarrolladores y contribuidores de código abierto. Es multiplataforma. Utiliza una estrategia de almacenamiento de filas para conseguir mejores respuestas en ambientes de grandes volúmenes. Presenta interfaces de comunicación con diferentes lenguajes, incluyendo C++ y PHP. Tiene altos niveles de escalabilidad, más rápido que otros gestores en la ejecución de transacciones. Tiene certificada la implementación ACID de sus transacciones. Presenta buenos niveles de administración de concurrencia de usuarios incluso en transacciones y consultas muy complejas. Se han desarrollado múltiples herramientas para ambientes de clusterización con PostgreSQL. Presenta mejores niveles de rendimiento en entornos Linux.

Librería Libpq.

Libpq es la librería interfaz para PostgreSQL desde C, es un conjunto de funciones que permiten a los programas cliente pasar las consultas al servidor PostgreSQL y recibir los resultados de estas consultas. Es también el motor subyacente para muchas otras interfaces de aplicación a PostgreSQL, incluyendo las programadas con C++, Perl, Python y Tcl. Libpq no es "thread-safety" pero permite la manipulación concurrente de sus objetos con la utilización de mecanismos de bloqueo de recursos como los semáforos en la programación multihilo. [38]

PgAdmin3.

PgAdmin3 es una completa herramienta multiplataforma de diseño y administración de Bases de Datos. Se distribuye libremente bajo los términos de la licencia Artistic Licence [29]. PgAdmin3 es la aplicación más popular, disponible para la administración de Bases de Datos PostgreSQL para la mayoría de Sistemas Operativos . Esta diseñado para responder a todas las necesidades de los

usuarios, desde la realización de consultas SQL hasta la elaboración de complejas sentencias. Ofrece fácil administración e incluye un editor de SQL para la ejecución de consultas.

Visual Paradigm-UML.

Visual Paradigm para UML es una de las herramientas para Ingeniería de Software de los proyectos más completas y fáciles de utilizar. Es multiplataforma y proporciona excelentes facilidades de operaciones con otras aplicaciones. Fue creada para el ciclo vital completo de desarrollo del software que lo automatiza y acelera, permitiendo la captura de requisitos, análisis, diseño e implementación. Proporciona herramientas para la generación del código, ingeniería inversa y generación de informes. Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de clases. Permite invertir código fuente de programas, archivos ejecutables y binarios en modelos UML al instante, creando de manera simple toda la documentación. Apoya los estándares más recientes de las notaciones de Java y de UML. Incorpora el soporte para trabajo en equipo, que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros.[39]

1.7. Conclusiones

En este capítulo se realizó un estudio de la persistencia de los datos en las aplicaciones que manejan grandes volúmenes de información, las características del acceso a datos en aplicaciones multihilos y alternativas para la distribución y alta disponibilidad de los datos en la actualidad. Además, se desarrolla el estudio de algunas de las tecnologías actuales que serán empleadas durante el desarrollo del software, mencionando sus características principales y el por qué su utilización para la realización del proyecto.

Capítulo 2

Motor de Persistencia de Datos para la plataforma Génesis

2.1. Introducción

La creación de un Motor de Persistencia de datos para Génesis ha garantizado mayor facilidad de configuración, mayor acoplamiento y mejoras en el diseño de la plataforma. En este capítulo se presenta el diseño de clases del Motor de Persistencia, su descripción y las principales características que le distinguen.

2.2. Diseño del Motor de Persistencia para la plataforma Génesis.

En la *figura 3.1* se muestra el diseño de clases del Motor de Persistencia de la plataforma Génesis con soporte para Bases de Datos PostgreSQL. Las clases persistentes heredan de la clase abstracta *IPersistente* que tiene un listado de campos que representa el listado de los campos de la tabla de la base de datos que le corresponde. Cada *IPersistente* tiene un Modo que representa el modo de inserción de los registros en la tabla. Puede ser Manual, cuando se especifica el valor de la clave primaria; Autoincrementable , cuando la clave primaria se genera de manera autoincremental y de Búsqueda cuando se genera por una función del sistema. Cada campo tiene una correspondencia de tipos de datos del sistema. Las clases persistentes tienen un apuntador a *IBaseDatos* que representa la base de datos

a la que está conectada. Las clases *IBaseDatos* e *IConexión* son clases abstractas de las que heredan las clases que representan Bases de Datos específicas y conexiones específicas como *CBDPostgreSQL* y *CConexionPostgres* para PostgreSql, respectivamente. Cuando se obtiene un conjunto de tuplas producto de una consulta realizada a partir de los métodos para recuperación de datos de las clases *IPersistente* e *IBaseDatos*, estas se devuelven en una estructura de tipo *CRegistroDatosMultiple*.

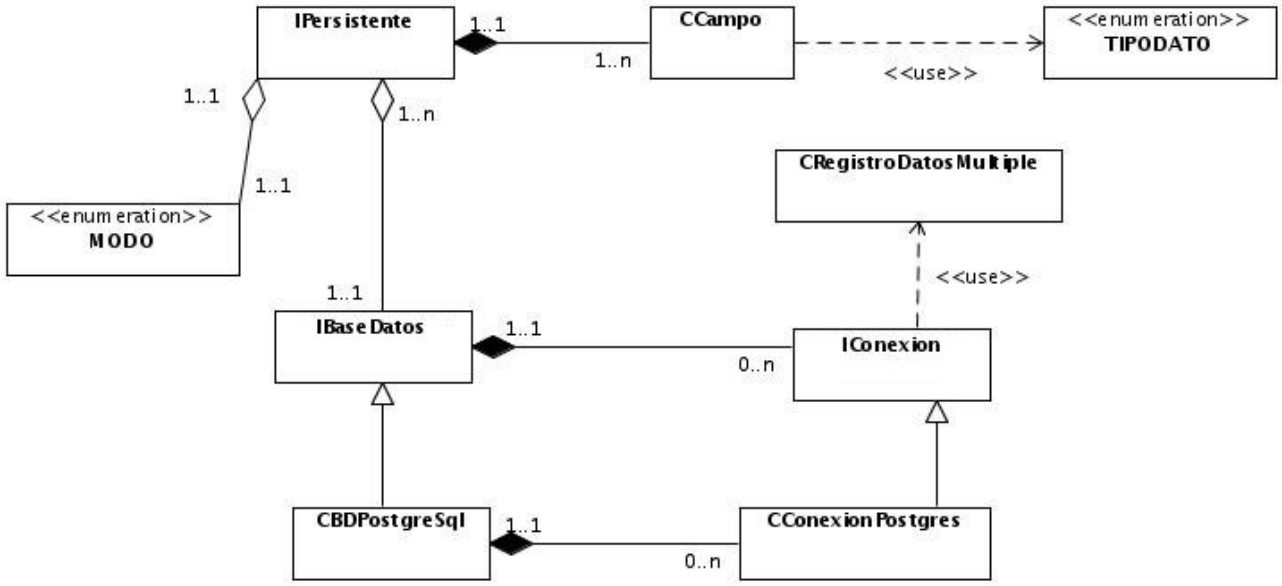

Figura 3.1 Diseño Clases del Motor de Persistencia Génesis con soporte para PostgreSQL

Descripción de las clases

Para un mayor entendimiento de sus funcionalidades se describen las clases del Motor de Persistencia a continuación.

Nombre de la clase: CCampo	
Descripción: Clase que representa los campos de las entidades persistentes.	
Atributos	**Tipo**
NombreCampo	STR
TipoDatosBD	STR
Dato	void*
Responsabilidades de la clase:	
Nombre:	CCampo()
Descripción:	Constructor de la clase sin parámetros.
Nombre:	CCampo(void* pDato, STR pNombreCampo, STR pTipoDatoBD)
Descripción:	Constructor de la clase con parámetros.
Nombre:	CCampo(const CCampo &pCampo)
Descripción:	Constructor de copia.
Nombre:	STR GetNombreCampo()
Descripción:	Devuelve el nombre del campo en la base de datos.
Nombre:	STR GetTipoDatoBD()
Descripción:	Devuelve el tipo de datos del campo en la base de datos.
Nombre:	TIPODATO GetTipoDatosSistema(IBaseDatos* pBD)
Descripción:	Devuelve el tipo de dato del sistema del campo.
Nombre:	void SetNombreCampo(STR pNombreCampo)
Descripción:	Establece el nombre del campo.
Nombre:	void SetTipoDatoBD(STR pTipoDatoBD)
Descripción:	Establece el tipo de datos de la base de datos del campo.
Nombre:	void SetDatoX(X pDato) (*)
Descripción:	Establece el valor del dato si es X.
Nombre:	X GetDatoX() (*)
Descripción:	Devuelve el valor del dato si es X.

Cuadro 2.1: Descripción de la Clase CCampo

Nombre de la clase: CRegistroDatosMultiple	
Descripción: Clase contenedora de las tuplas resultados de una transacción.	
Atributos	**Tipo**
TipoDatosSis	vector<TIPODATO>
NombreCampos	vector<STR>
Valores	vector< vector<void*> >
CantidadCampos	ULI32
CantidadRegistros	ULI32
Responsabilidades de la clase:	
Nombre:	CRegistroDatosMultiple()
Descripción:	Constructor de la clase sin parámetros
Nombre:	CRegistroDatosMultiple(ULI32 pCantidadRegistros)
Descripción:	Constructor de la clase a partir de la Cantidad de Registros.
Nombre:	CRegistroDatosMultiple(CRegistroDatosMultiple& pReg)
Descripción:	Constructor de copia.
Nombre:	ULI32 GetCantidadCampos()
Descripción:	Devuelve la cantidad de campos asociados al registro.
Nombre:	ULI32 GetCantidadRegistros()
Descripción:	Devuelve la cantidad de registros o tuplas.
Nombre:	vector<void*> GetValoresPorFila(ULI32 pFila)
Descripción:	Devuelve una tupla de valores.
Nombre:	TIPODATO GetTipoDatosSistema(ULI32 pPosCampo)
Descripción:	Devuelve el Tipo de datos del sistema de un campo dada una posición.
Nombre:	STR GetNombreCampo(ULI32 pPosCampo)
Descripción:	Devuelve el nombre del campo dada una posición.
Nombre:	void SetTipoDatosSistema(ULI32 pposición, TIPODATO pTipoDatos)
Descripción:	Establece el Tipo de datos del sistema de registro.
Nombre:	void SetNombreCampos(ULI32 pposición, STR pCampo)
Descripción:	Establece el nombre del campo dada una posición.
Nombre:	void SetCantidadRegistros(ULI32 pCantidadRegistros)
Descripción:	Establece la cantidad de tuplas del CRegistroMultiple.
Nombre:	void AdicionarX(X pValor, STR pNombreCampo, ULI32 pFila) (*)
Descripción:	Adiciona un valor de tipo X dado el nombre del campo y la fila.
Nombre:	X ObtenerXporPOS(ULI32 pposición, ULI32 pFila) (*)
Descripción:	Devuelve el valor dada la posición del campo y la fila si es de tipo X.
Nombre:	X ObtenerXporNombre(STR pNombreCampo, ULI32 pFila) (*)
Descripción:	Devuelve el valor dado el nombre del campo y la fila si es de tipo X.

Cuadro 2.2: Descripción de la Clase CRegistroDatosMultiple

Nombre de la clase: IConexion.	
Descripción: Clase abstracta que representa la conexión a una BD.	
Responsabilidades de la clase:	
Nombre:	IConexion()
Descripción:	Constructor de la clase.
Nombre:	virtual void Conectar(STR pNomServidor, STR pBDatos,STR pUsuario,STR pContrasenna)=0
Descripción:	Establece la conexión.
Nombre:	virtual STR GetErrorMasReciente()=0
Descripción:	Devuelve el error más reciente arrojado por el gestor de Bases de Datos.
Nombre:	virtual void SetConexion(STR pNomServidor,STR pBDatos, STR pUsuario, STR pContrasenna)=0
Descripción:	Restablece la conexión con los parámetros nuevos.
Nombre:	virtual void CerrarConexion()= 0;
Descripción:	Cierra la conexión actual a la Base de Datos.
Nombre:	virtual void ReiniciarConexion()= 0;
Descripción:	Reinicia la conexión actual a la Base de Datos.
Nombre:	virtual void EjecutarComando(STR pSql) =0;
Descripción:	Ejecuta la sentencia pSql. Utilizada para la Ejecución de operaciones de escritura.
Nombre:	virtual CRegistroDatosMultiple* EjecutarConsulta(STR pSql)=0;
Descripción:	Ejecuta la sentencia pSql. Utilizada para la ejecutar operaciones de lectura.

Cuadro 2.3: Descripción de la Clase IConexion

Nombre de la clase: CConexionPostgres.	
Descripción: Clase para la conexión a SGBD PostgreSQL. Hereda de IConexion.	
Responsabilidades de la clase:	
Nombre:	CConexionPostgres();
Descripción:	Constructor de la clase.
Nombre:	STR GetErrorMasReciente();
Descripción:	Devuelve el error más reciente arrojado por la Base de Datos PostgreSQL
Nombre:	void Conectar(STR pNombreServidor, STR pNombreBDatos,STR pUsuario,STR pContrasenna);
Descripción:	Establece una conexión a una Base de Datos PostgreSQL.
Nombre:	void SetConexion(STR pNombreServidor,STR pNombreBDatos, STR pUsuario,STR pContrasenna);
Descripción:	Restablece la conexión con los parámetros nuevos a una Base de Datos PostgreSQL.
Nombre:	void CerrarConexion();
Descripción:	Cierra la conexión actual a la Base de Datos PostgreSQL.
Nombre:	void ReiniciarConexion();
Descripción:	Reinicia la conexión actual a una Base de Datos PostgreSQL.
Nombre:	void EjecutarComando(STR pSql);
Descripción:	Ejecuta la sentencia pSql. Utilizada para la Ejecución de operaciones de escritura.
Nombre:	CRegistroDatosMultiple* EjecutarConsulta(STR pSql);
Descripción:	Ejecuta la sentencia pSql. Utilizada para la ejecutar operaciones de lectura.

Cuadro 2.4: Descripción de la Clase ConexionPostgres.

Nombre de la clase: CBDPostgreSQL.	
Descripción: Clase que hereda de IBaseDatos. Crea objetos de tipo CConexionPostgres.	
Atributo	Tipo de Dato
NombreServidor	STR
NombrBDatos	STR
Usuario	STR
Contrasenna	STR
PuertoBD	STR
Responsabilidades de la clase:	
Nombre:	CBDPostgreSQL();
Descripción:	Constructor sin parámetros
Nombre:	CBDPostgreSQL(CConexionPostgres* pConexion);
Descripción:	Constructor con parámetros.
Nombre:	~CBDPostgreSQL();
Descripción:	Destructor de la clase.
Nombre:	TIPODATO GetTipoSistema(LI32 pTipoDatoBD);
Descripción:	Devuelve el tipo de Datos del Sistema dado el identificador del tipo de datos de una BD PostgreSQL.
Nombre:	TIPODATO GetTipoSistema(STR pTipoDatoBD);
Descripción:	Devuelve el tipo de Datos del Sistema dado el nombre del tipo de datos de una BD PostgreSQL.
Nombre:	IBaseDatos* Clonar();
Descripción:	Devuelve una copia exacta del objeto.

Cuadro 2.5: Descripción Clase CBDPostgreSQL

Nombre de la clase: IBaseDatos.	
Descripción: Clase abstracta que contiene la información de una BD en el Sistema.	
Atributo	**Tipo**
NombreServidor	STR
NombrBDatos	STR
Usuario	STR
Contrasenna	STR
PuertoBD	STR
ConexionPtr	IConexion*
Responsabilidades de la clase:	
Nombre:	IBaseDatos()
Descripción:	Constructor de la clase.
Nombre:	~IBaseDatos()
Descripción:	Destructor de la clase
Nombre:	void ConectarBD()
Descripción:	Invoca al método Conectar de ConexionPtr que abre una conexión a la BD
Nombre:	void DesconectarBD();
Descripción:	Cierra la conexión existente a la base de datos.
Nombre:	virtual TIPODATO GetTipoSistema(LI32 pTipoDatoBD) = 0;
Descripción:	Devuelve el tipo de Datos del Sistema dado el identificador del tipo de datos en la Base de Datos
Nombre:	virtual TIPODATO GetTipoSistema(STR pTipoDatoBD) = 0;
Descripción:	Devuelve el tipo de Datos del Sistema dado el nombre del tipo de datos en la Base de Datos
Nombre:	virtual IBaseDatos* Clonar()=0;
Descripción:	Devuelve una copia de la Base de Datos.
Nombre:	void SetNombreBDatos(STR pNombreBDatos)
Descripción:	Establece el Nombre de la Base de Datos.
Nombre:	void SetUsuario(STR pUsuario);
Descripción:	Establece el Usuario de la Base de Datos.
Nombre:	void SetPtoServidorBD(USI16 pPtoServidorBD)

Descripción:	Establece el puerto de conexión a la Base de Datos.
Nombre:	void SetNombreServidor(STR pNombreServidor)
Descripción:	Establece el Nombre del Servidor.
Nombre:	void SetContrasenna(STR pContrasenna)
Descripción:	Establece la Contraseña.
Nombre:	CRegistroDatosMultiple* EjecutarConsulta(STR pSql)
Descripción:	Envia la consulta 'pSql' a ConexionPtr para su ejecución.
Nombre:	void EjecutarComando(STR pSql)
Descripción:	Envia el comando 'pSql' a ConexionPtr para su ejecución.

Cuadro 2.6: Descripción Clase clase abstracta IBaseDatos

Nombre de la clase: IPersistente.	
Descripción: Clase abstracta de la que heredan todas las clases persistentes.	
Atributos (**)	**Tipo**
BD	IBaseDatos*
Campos	vector<CCampo>
CantCampos	ULI32
Indicellaves	vector<USI16>
ULlaveConsultada	vector<STR>
Modo	MODO
Tabla	STR
Responsabilidades de la clase:	
Nombre:	IPersistente()
Descripción:	Constructor de la clase sin parámetros.
Nombre:	B8 Iterar()
Descripción:	Devuelve verdadero en caso que pueda seguir iterando, false en caso contrario.
Nombre:	void SetNombreTabla(STR pNombre)
Descripción:	Establece el nombre de la tabla al que corresponde la clase.
Nombre:	void Conectar(IBaseDatos* pBD)
Descripción:	Conecta la clase persistente a la BD.
Nombre:	void Desconectar()
Descripción:	Desconecta la clase persistente a la BD.
Nombre:	void TieneCampoX(X &pCont, STR pNombre, STR pTipoBD, B8 pEsLlave) (*)

Descripción:	Añade a la clase persistente un campo de tipo X.
Nombre:	void Insertar()
Descripción:	Inserta los valores de la clase persistente en su tabla.
Nombre:	void Actualizar()
Descripción:	Actualiza el registro con los valores nuevos que contiene.
Nombre:	void ActualizarTodos()
Descripción:	Actualiza todos los registros de la tabla con los valores del actual.
Nombre:	void Eliminar()
Descripción:	Elimina el registro correspondiente a la clave.
Nombre:	void EliminarTodos()
Descripción:	Elimina todos los registros de la tabla.
Nombre:	CRegistroDatosMultiple* SeleccionarIterando(ULI32 pCantidad)
Descripción:	Selecciona una cantidad determinada (pCantidad) de registros de la tabla.
Nombre:	CRegistroDatosMultiple* SeleccionarIterandoDonde(ULI32 pCantidad, STR pSQL)
Descripción:	Selecciona una cantidad determinada de registros de la tabla dada una condición.

Cuadro 2.7: Descripción de la Clase IPersistente.

Las definiciones enumerativas para TIPODATO y MODO se refieren a los valores de los tipos de datos soportados por el sistema y los modos de inserción de los registros en las clases persistentes respectivamente.

Las funciones marcadas (*) se describen de manera genérica, puesto que se repite en la clase correspondiente la misma funcionalidad para cada tipo de dato del sistema[1] representado por X.

2.3. Características del Motor de Persistencia de la plataforma Génesis.

1. **Mapeo de clases a tablas.** Establece una correspondencia entre las tablas de las Bases de Datos y sus campos a clases del diseño orientado a objetos y atributos respectivamente. Todas las clases que mapean las tablas de las Bases de Datos, heredan de *IPersistente*, como se muestra en la *figura 2.1.*

[1] Consultar los tipos de datos soportados por el Motor de Persistencia en el la *sección 2.3(7)*

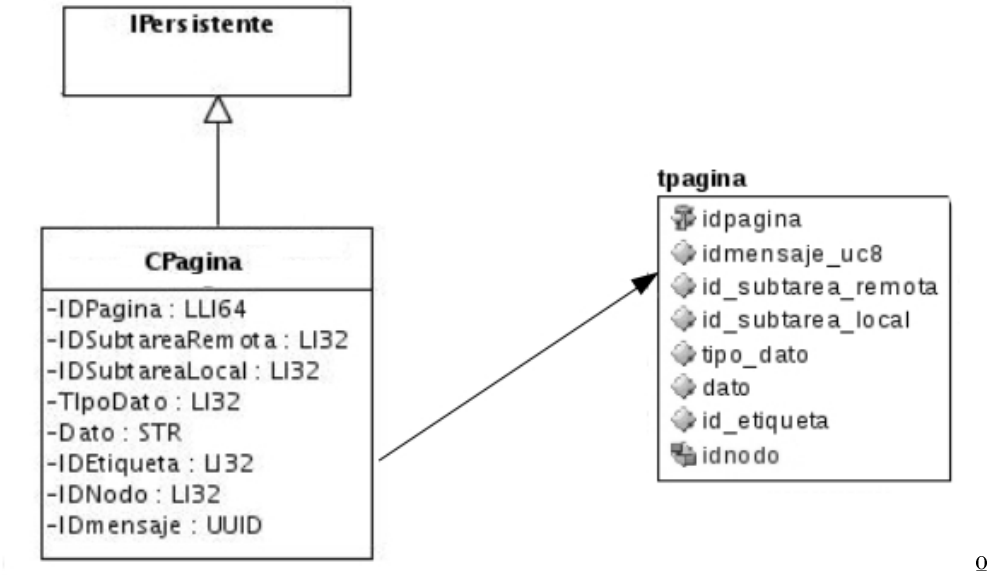

Figura 2.1: Correspondencia entre la clase persistente CPagina y la relación tpagina.

En el constructor de la clase persistente que se crea se establece la correspondencia con la tabla y los atributos del modelo relacional de la siguiente manera *(figura 2.2)*:

```
CPagina::CPagina()
{
    SetNombreTabla("public.\"tpagina\"");
    TieneCamposLLI64(IDPagina, "idpagina", "bigint", true);
    TieneCamposLI32(IDSubtareaLocal,"id_subtarea_local", "integer");
    TieneCamposLI32(IDSubtareaRemota,"id_subtarea_remota", "integer");
    TieneCamposLI32(TipoDato,"tipo_dato", "integer");
    TieneCamposSTR(Dato, "dato", "varchar");
    TieneCamposLI32(IDEtiqueta, "id_etiqueta","integer");
    TieneCamposLI32(IDNodo,"idnodo", "integer");
    TieneCamposUUID(IDMensaje, "idmensaje_uc8", "uuid");
}
```

Figura 2.2: Proceso de mapeo de clases persistentes.

2. **Arquitectura flexible para añadir soporte a varios tipos de Bases de Datos.** En la actualidad el Motor de Persistencia solo se ha probado para el SGBD PostgreSQL. Para añadir soporte para conexión a un tipo de base de datos determinado, se debe heredar de las clases abstractas *IConexión* e *IBaseDatos* y redefinir sus métodos de acuerdo a las especificaciones de las *tablas 2.3* y *2.5* respectivamente. Estas clases se compilan en una librería que exporta un método *IBaseDatos* GetBaseDatos()*.

   Las librerías tienen una correspondencia única con los tipos de Bases de Datos que soporta el Motor de Persistencia. Exportan los objetos que actúan como intermediarios para la manipulación de los SGBD que representan.

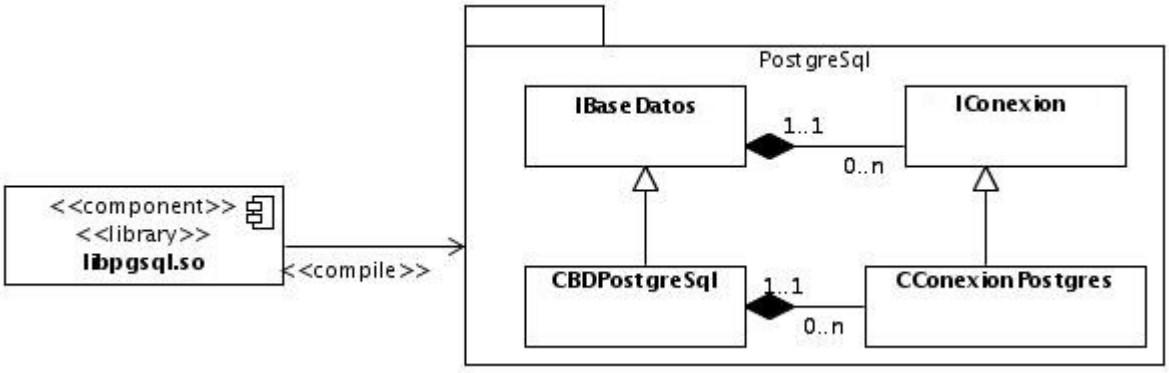

Figura 2.3: Clases de la librería para conexión a SGBD PostgreSQL.

3. **Fácil configuración de acceso a varias Bases de Datos.** Los parámetros de configuración de las Bases de Datos que serán consultadas por la plataforma deben aparecer en el fichero de configuración génesis.conf[2] como se muestra en la *figura 2.3.*

[2]El fichero génesis.conf se localiza para plataformas Linux en /etc/génesis.conf

```
#En esta sección se describen los parámetros de configuración de las Bases de Datos
# a las que se puede conectar la plataforma.

basedatos
{
  idbd genesis { nombreservidorbd= eipad5.uci.cu  puerto = 5432 nombrebd= Genesis usuariobd= xxxxx  password= xxxxxxx
                 driver= libpgsql.so }

  idbd aduana { nombreservidorbd= eipad4.uci.cu  puerto = 5432  nombrebd= Aduana  usuariobd= xxxxx password= xxxxxx
                driver= libpgsql.so }
}
```

Figura 2.4: Configuración de Bases de Datos consultadas por Génesis.

Una vez que el MPPP inicia sus servicios el Lector de Parámetros de Configuración invoca las librerías configuradas y adiciona a los objetos que estas exportan los parámetros especificados. A través del Lector de Parámetros de Configuración se obtiene acceso a dichos objetos por el nombre del "idbd" que tiene la base de datos en el fichero, por ejemplo:

*IBaseDatos *GenesisBDPtr = CLectorparámetrosConfiguracion::GetLector()->GetBD("genesis");*

4. **Las clases persistentes realizan su propio CRUD.** Con la llamada a los métodos *void Insertar(), void Actualizar(), void Eliminar(), void Seleccionar()* y sus variantes descritas en la *tabla 2.7* cada objeto de la clase puede insertar sus valores en la tabla a la que pertenece, actualizar o eliminar el registro que le pertenece a su clave primaria y seleccionar de manera iterativa los registros de la tabla correspondiente.

5. **Recuperación de objetos.** Permite la recuperación de una lista de objetos persistidos a través de la ejecución de consultas en lenguaje SQL, no mediante un lenguaje de consulta especial entre objetos, aspecto que aporta eficiencia en el desempeño de los módulos de cómputo. Los resultados se devuelven en la estructura *CRegistroMultipleDatos*. Los datos pueden obtenerse por la posición del campo en la consulta o por el nombre, se aconseja acceder por la posición que tiene un orden constante a diferencia del nombre que aunque pequeño introduce un tiempo variable en dependencia de la cantidad de campos. En la *tabla 2.2* se describe la clase *CRegistroMultipleDatos* para mejor comprensión de sus funcionalidades.

6. **Regula la carga de las consultas de selección.** Debido a las características de la plataforma de manipular gran cantidad de datos, el Motor de Persistencia permite la recuperación de objetos regulando la carga, es decir, extrae una cantidad regulable de objetos de manera iterativa. Esta forma permite seleccionar todos los registros de la base de datos sin cargarlos en memoria al mismo tiempo, a través de los métodos *CRegistroDatosMultiple* SeleccionarIterando(ULI32 pCantidad)* y *B8 Iterar()* de la clase *IPersistente.*

7. **Tipos de datos soportados.** Una de las principales características de los motores de persistencia es que establecen una correspondencia entre los tipos de datos de las Bases de Datos y los tipos de datos que estos soportan. Cuando se añade una librería para conexión a un tipo de base de datos, es responsabilidad del programador establecer soporte para los siguientes tipos de datos de la plataforma:

Tipo	Descripción
C8	Caracter con signo de 8 bits.
UC8	Caracter sin signo de 8 bits.
I32	Entero con signo de 32 bits.
UI32	Entero sin signo de 32 bits.
LI32	Entero largo con signo de 32 bits.
ULI32	Entero largo sin signo de 32 bits.
LLI64	Entero con signo de 64 bits.
SI16	Entero corto con signo de 16 bits.
USI16	Entero corto sin signo de 16 bits.
F32	Flotante de precisión simple de 32 bits.
D64	Flotante de precisión doble de 64 bits.
LD96	Flotante de precisión doble de 96 bits.
STR	Cadena de caracteres.
B8	Booleano de 8 bits.
UUID	Arreglo de caracteres sin signo de 128 bits.

Cuadro 2.8: Descripción de los tipos de datos soportados por el Motor de Persistencia.

2.4. Tratamiento de errores

El tratamiento de errores en el Motor de Persistencia está estrechamente relacionado al tratamiento de errores en la plataforma Génesis. Se basa en el lanzamiento de excepciones que nos ofrece el lenguaje

C++. A partir de un conjunto de errores previamente identificados, se elaboró un diseño de clases correspondiente a los tipos de errores de manera que existe una para cada tipo de error. Este método estandariza el tratamiento de errores de manera que el resto de las partes de la plataforma reciben las mismas excepciones independientemente del tipo de base de datos a la que se conecta la aplicación. A continuación se muestran el diseño de clases para la gestión de errores de Bases de Datos y en la *tabla 2.9* muestra la relación de estas con los tipos de errores a los que corresponden.

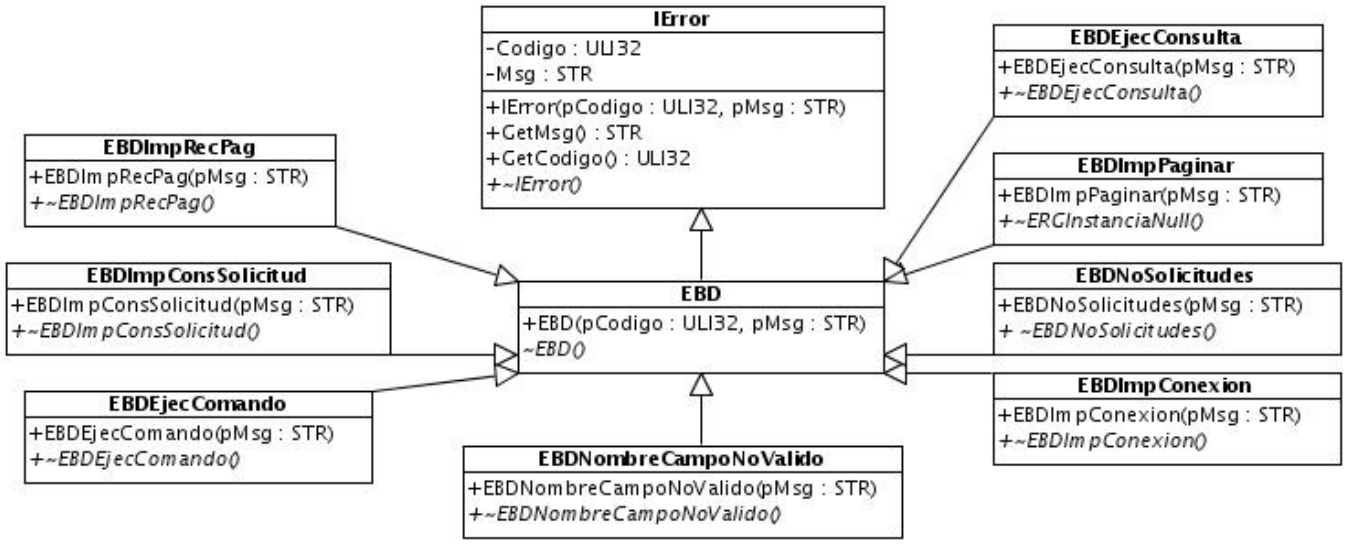

Figura 2.5: Clases para el tratamiento de errores de Bases de Datos.

Clase Error (Identificador)	EBDImpPaginar (EBD_IMP_PAGINAR)
Descripción	No se pudo paginar en el Buffer de Mensajes.
Clase Error (Identificador)	EBDImpRecPag (EBD_IMP_REC_PAGINA)
Descripción	No se pudo recuperar el paginado en el Buffer de Mensajes.
Clase Error (Identificador)	EBDNoSolicitudes(EBD_NO_SOLICITUDES)
Descripción	No hay solicitudes para procesar en la Base de datos.
Clase Error (Identificador)	EBDImpConsSolicitud (EBD_IMP_CONS_SOLICITUD)
Descripción	No se pudieron consultar las solicitudes.
Clase Error (Identificador)	EBDImpConexion (EBD_IMP_CONEXION)
Descripción	No se pudo establecer la conexión a la BD.
Clase Error (Identificador)	EBDEjecComando (EBD_EJEC_COMANDO)
Descripción	No se pudo ejecutar el comando en la BD.
Clase Error (Identificador)	EBDEjecConsulta (EBD_EJEC_CONSULTA)
Descripción	No se pudo ejecutar la consulta en la BD.
Clase Error (Identificador)	EBDCampoNoValido (EBD_CAMPO_NO_VALIDO)
Descripción	No existe un campo en la tabla con ese nombre.

Cuadro 2.9: Descripción de errores de Bases de Datos y clases que le corresponden.

2.5. Conclusiones

En este capítulo se presentaron las especificaciones del Motor de Persistencia de la plataforma Génesis, proporcionando un conocimiento básico del mismo. El estado actual del componente satisface los requerimientos de la plataforma en la fase de desarrollo en que se encuentra, aunque se pretende añadir nuevas funcionalidades y optimizar las existentes. Este componente contribuye a garantizar una de las principales potencialidades de la plataforma que es el acceso a los datos.

Capítulo 3

Capa de Acceso a Datos de la plataforma Génesis.

3.1. Introducción

La estructura de almacenamiento de datos de la plataforma Génesis interactúa con varias aplicaciones, los Motores de Procesamiento de Peticiones Paralelas en modo Cliente (MPPC) o Servidor (MPPS) para el procesamiento de las solicitudes de cómputo y la Aplicación de Administración y Monitorización, por lo que ha sido desarrollada con el objetivo de satisfacer los requerimientos de las mismas permitiendo el almacenamiento de los datos, el reconocimiento de su contenido y la recuperación de la información, así como un diseño flexible que medie entre ambos entornos de software. En este capítulo se describe la capa de acceso a datos de la plataforma Génesis de las aplicaciones MPPPC y MPPPS implementada con el Motor de Persistencia, se muestran los requerimientos en los que se basa el diseño de clases persistentes y de la BD. Se describe parte del diseño de los componentes de estas aplicaciones que acceden a la BD y se documentan las especificaciones para la adición de módulos de cómputo.

3.2. Requerimientos de la plataforma Génesis.

A continuación se muestran los requerimientos de las aplicaciones MPPPS y MPPPC en los que estuvo basado el diseño para la persistencia de datos.

3.2.1. Requerimientos funcionales.

1. Procesar solicitudes : El MPPPS manipula las solicitudes de cálculos.

   a) Seleccionar la próxima solicitud de cálculo cuya complejidad sea menor que la carga que el MPPPS es capaz de procesar y esté en espera de procesamiento.

   b) Cambiar el estado de procesamiento de una solicitud determinada.

   c) Calcular complejidad total de las solicitudes actuales en procesamiento.

2. Notificar registros del sistema. Cada nodo inserta información del sistema en forma de registros.

3. Paginar mensajes del Buffer de Mensajes cuando este ha llegado al límite de la capacidad de almacenamiento.

   a) Insertar nuevos mensajes de un nodo.

   b) Devolver a partir de la información de un mensaje, el dato asociado a este.

   c) Eliminar un mensaje a partir de su identificador.

   d) Eliminar todos los mensajes de un nodo determinado.

3.2.2. Requerimientos no funcionales.

1. Software: Se necesita de las instalaciones de un clúster de base de datos PostgreSQL 8.3 en plataforma Linux Debian 8.6.26-2 y PgCluster 1.9 con Heartbeat.

2. Restricciones en el diseño y la implementación: Las aplicaciones MPPPS y MPPPC están implementadas en lenguaje de programación C++, utiliza las librerías libpqxx3.0, uuid1.5.

3. Disponibilidad: Los usuarios autorizados se les garantizará el acceso a la información.

3.3. Diseño de Clases Persistentes.

El modelo de clases persistentes que se identifican a partir de las clases del diseño que responden a los requerimientos anteriores y su descripción se muestran a continuación:

3.3.1. Diagrama de clases persistentes.

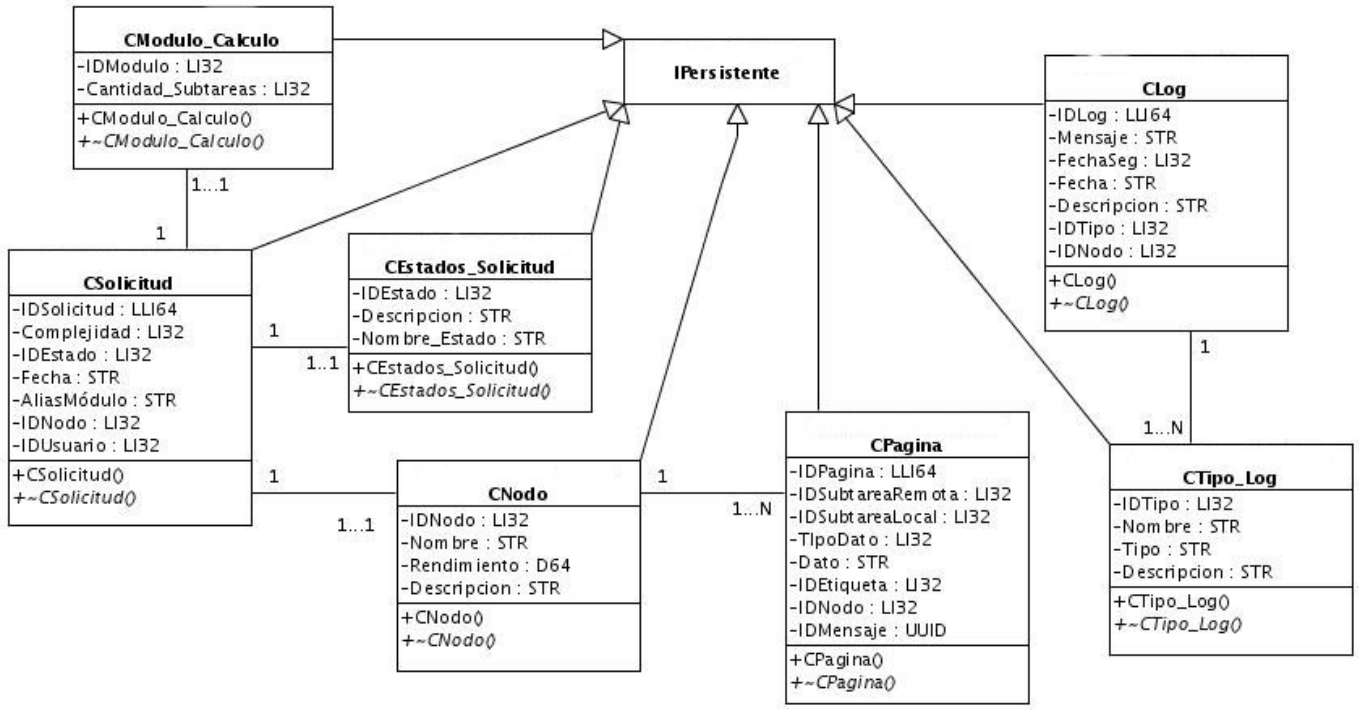

Figura 3.1: Diagrama de Clases Persistentes del Diseño.

3.3.2. Descripción de las clases persistentes.

Nombre de la clase: CModulo_Calculo	
Descripción: Clase persistente que representa a los módulos de cálculo. Hereda las funcionalidades descritas en la clase IPersistente.	
Atributos	**Tipo**
idmodulo	LI32
cantidad_subtareas	LI32
Responsabilidades de la clase:	
Nombre:	CModulo_Calculo()
Descripción:	Constructor de la clase sin parámetros.
Nombre:	~CModulo_Calculo()
Descripción:	Destructor de la clase.

Cuadro 3.1: Descripción de la Clase CModulo_Calculo

Nombre de la clase: CSolicitud	
Descripción: Clase persistente que representa las solicitudes. Hereda las funcionalidades descritas en la clase IPersistente.	
Atributos	**Tipo**
IDSolicitud	LLI64
Complejidad	LI32
IDEstado	LI32
Fecha	STR
AliasModulo	STR
IDNodo	LI32
IDUsuario	LI32
Responsabilidades de la clase:	
Nombre:	CSolicitud()
Descripción:	Constructor de la clase sin parámetros.
Nombre:	~CSolicitud()
Descripción:	Destructor de la clase.

Cuadro 3.2: Descripción de la Clase CSolicitud

Nombre de la clase: CEstados_Solicitud	
Descripción: Clase persistente que representa los estados de las solicitudes. Hereda las funcionalidades descritas en la clase IPersistente.	
Atributos	**Tipo**
IDEstado	LI32
Descripcion	STR
Nombre_Estado	STR
Responsabilidades de la clase:	
Nombre:	CEstados_Solicitud()
Descripción:	Constructor de la clase sin parámetros.
Nombre:	~CEstados_Solicitud()
Descripción:	Destructor de la clase.

Cuadro 3.3: Descripción de la Clase CEstados_Solicitud

Nombre de la clase: CNodo	
Descripción: Clase persistente que representa a los nodos. Hereda las funcionalidades descritas en la clase IPersistente.	
Atributos	**Tipo**
IDNodo	LI32
Nombre	STR
Rendimiento	D64
Descripcion	STR
Responsabilidades de la clase:	
Nombre:	CNodo()
Descripción:	Constructor de la clase sin parámetros.
Nombre:	~CNodo()
Descripción:	Destructor de la clase.

Cuadro 3.4: Descripción de la Clase CNodo

Nombre de la clase: CPagina	
Descripción: Clase persistente que representa los mensajes que pueden paginarse. Hereda las funcionalidades descritas en la clase IPersistente.	
Atributos	**Tipo**
IDPagina	LLI64
IDSubtareaRemota	LI32
IDSubtareaLocal	LI32
TipoDato	LI32
Dato	STR
IDEtiqueta	LI32
IDNodo	LI32
IDMensaje	UUID
Responsabilidades de la clase:	
Nombre:	CPagina()
Descripción:	Constructor de la clase sin parámetros.
Nombre:	~CPagina()
Descripción:	Destructor de la clase.

Cuadro 3.5: Descripción de la Clase CPagina

Nombre de la clase: CLog	
Descripción: Clase persistente que representa a los registros. Hereda las funcionalidades descritas en la clase IPersistente.	
Atributos	**Tipo**
IDLog	LLI64
Mensaje	STR
Fecha	STR
FechaSeg	LI32
Descripcion	STR
IDNodo	LI32
IDTipo	LI32
Responsabilidades de la clase:	
Nombre:	CLog()
Descripción:	Constructor de la clase sin parámetros.
Nombre:	~CLog()
Descripción:	Destructor de la clase.

Cuadro 3.6: Descripción de la Clase CLog

Nombre de la clase: CTipo_Log	
Descripción: Clase persistente que representa a los tipos de registro. Hereda las funcionalidades descritas en la clase IPersistente.	
Atributos	**Tipo**
IDTipo	LI32
Nombre	STR
Tipo	STR
Descripcion	STR
Responsabilidades de la clase:	
Nombre:	CTipo_Log()
Descripción:	Constructor de la clase sin parámetros.
Nombre:	~CTipo_Log()
Descripción:	Destructor de la clase.

Cuadro 3.7: Descripción de la Clase CTipo_Log

3.4. Diseño de la Base de Datos

3.4.1. Diagrama Entidad Relación de la BD.

La siguiente figura representa el diagrama entidad relación correspondiente a la Base de Datos de la plataforma Génesis.

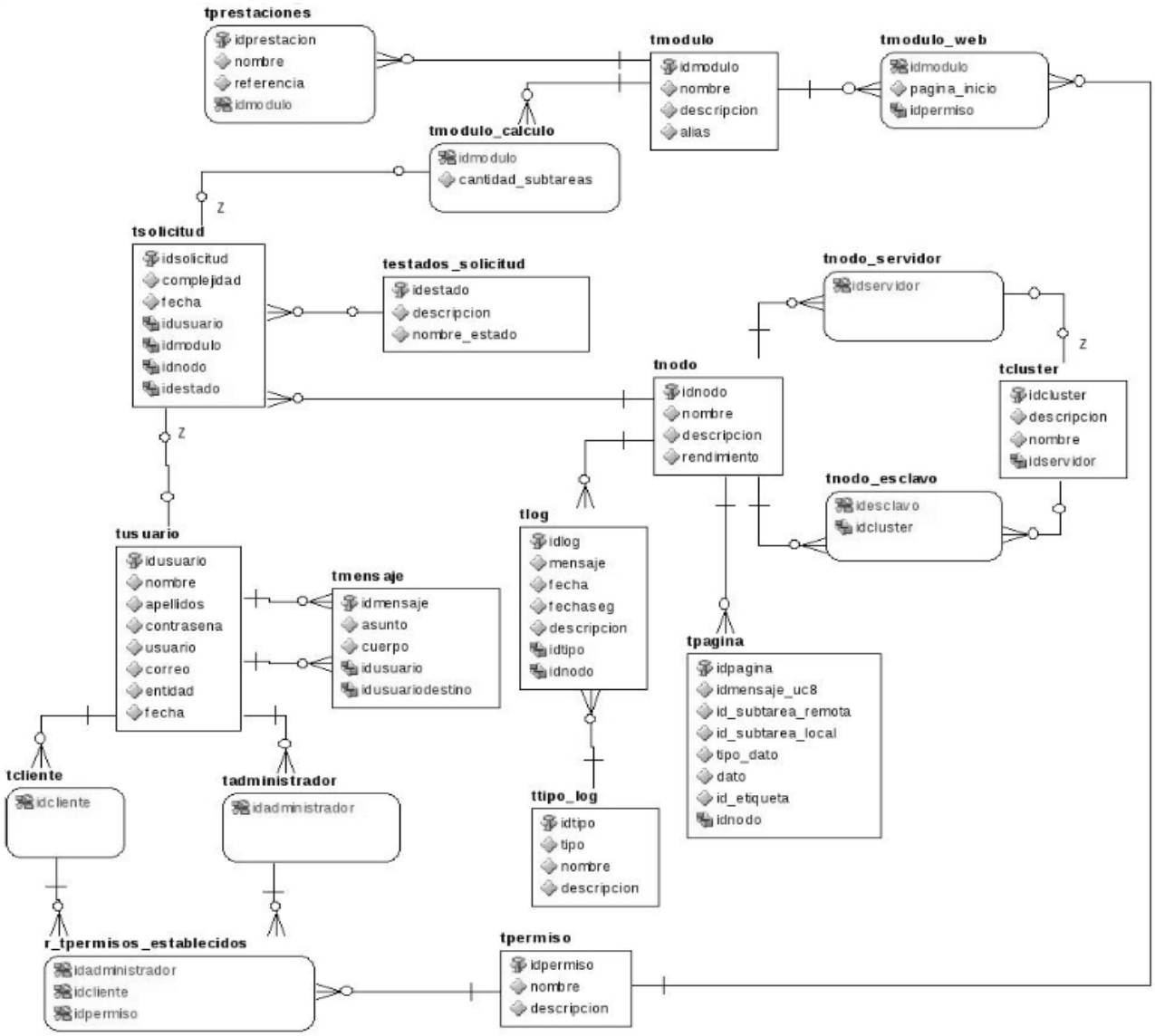

Figura 3.2: Modelo Entidad Relación de la plataforma Génesis.

A continuación se muestra dicho modelo dividido en 2 submodelos para una mejor comprensión.

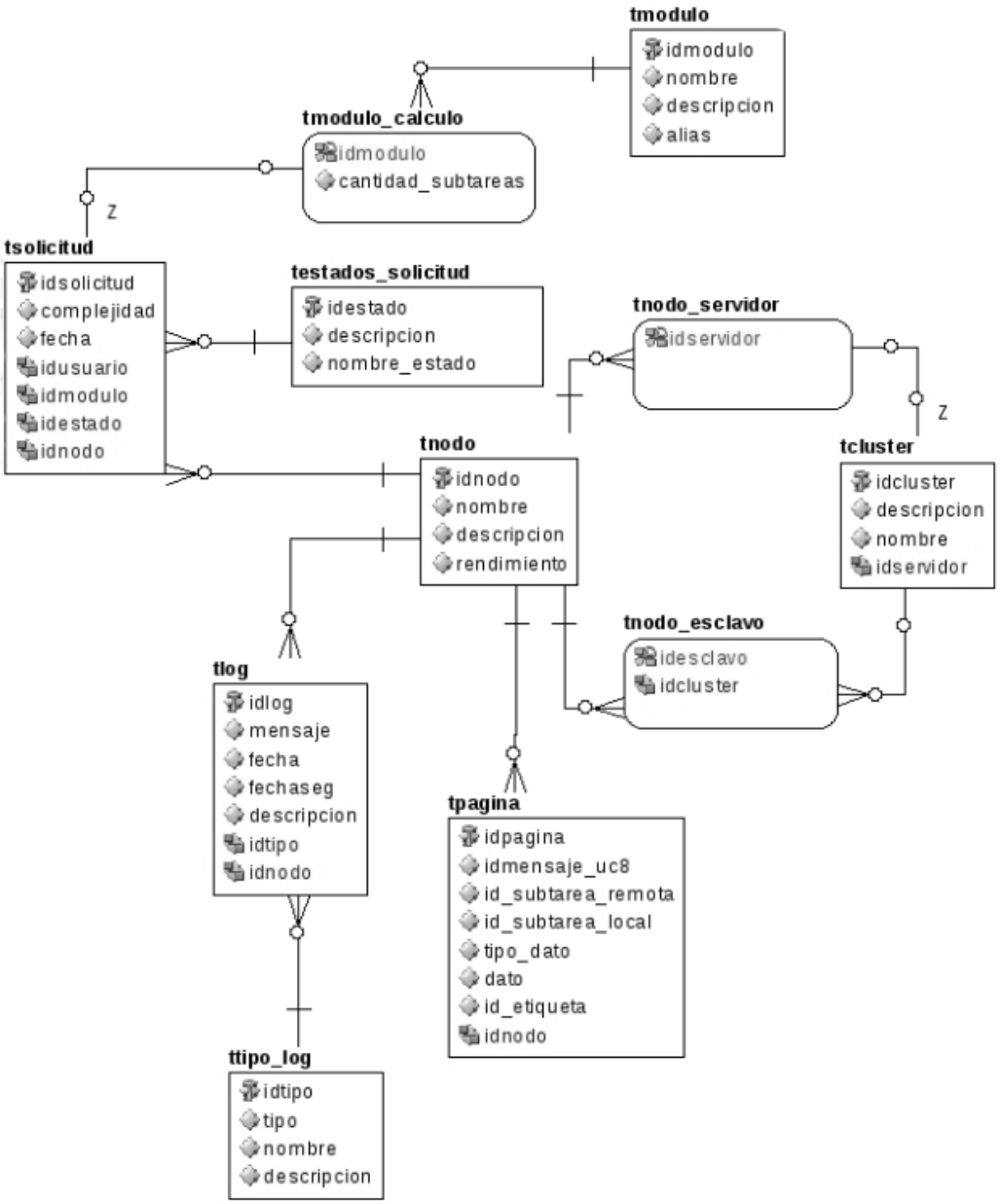

Figura 3.3: Submodelo Entidad Relación para las aplicaciones MPPS y MPPC

Descripción:

El submodelo de datos anterior corresponde a la siguiente descripción. Una vez insertada una solicitud de cómputo en la tabla tsolicitud, el MPPS lee sus datos para comenzar el procesamiento. Los módulos de cálculo son los destinados específicamente para el cómputo. Es por eso que están directamente relacionados con las solicitudes a través del idmodulo.

Cada solicitud tiene un estado de procesamiento. Estos estados están almacenados en la tabla testados_solicitud y se relacionan a partir del idestado_solicitud. Inicialmente el estado de las solicitudes es "En espera" y varía en dependencia de los niveles de procesamiento en que esta se encuentre hasta que finaliza. El MPPPS es el encargado de actualizar este dato.

Una vez que la subtarea máster de la solicitud ha sido asignada a un nodo específico del clúster este dato es actualizado en la tabla tsolicitud (relacionado a través del idnodo) y el estado de la solicitud es actualizado "En procesamiento".

Cada nodo del clúster de procesamiento ya sea en modo cliente o servidor realizan paginado, cuando la cantidad de mensajes que se envían entre los nodos sobrepasa el límite del Buffer de Mensajes de cada nodo, este los guarda en la base de datos, y cuando son solicitados los extrae. La información de estos mensajes se almacenan en la tabla tpagina. Un nodo puede paginar muchos mensajes y cada mensaje corresponde a un único nodo.

Cada nodo del clúster de procesamiento tiene además, su reporte de registros, información que se almacena en la tabla tlog, cada registro es de un tipo específico (almacenados en la tabla ttipo_log y relacionados por el atributo idtipo), que lo clasifica en dependencia de la magnitud que tiene la información que trae. Un nodo puede reportar muchos registros y cada registro es emitido por un único nodo.

En la tabla tcluster se almacena la información de los clústers de procesamiento que manipula la plataforma. En la tabla tnodo se almacena la información de todos los nodos incorporados al clúster de procesamiento. Los nodos pueden ser de dos tipos: nodo servidor o nodo esclavo, en dependencia de la aplicación MPPP que utilizan. Un clúster está compuesto por al menos un nodo esclavo y tiene un nodo servidor relacionados a través de los identificadores de los nodos (idservidor e idesclavo).

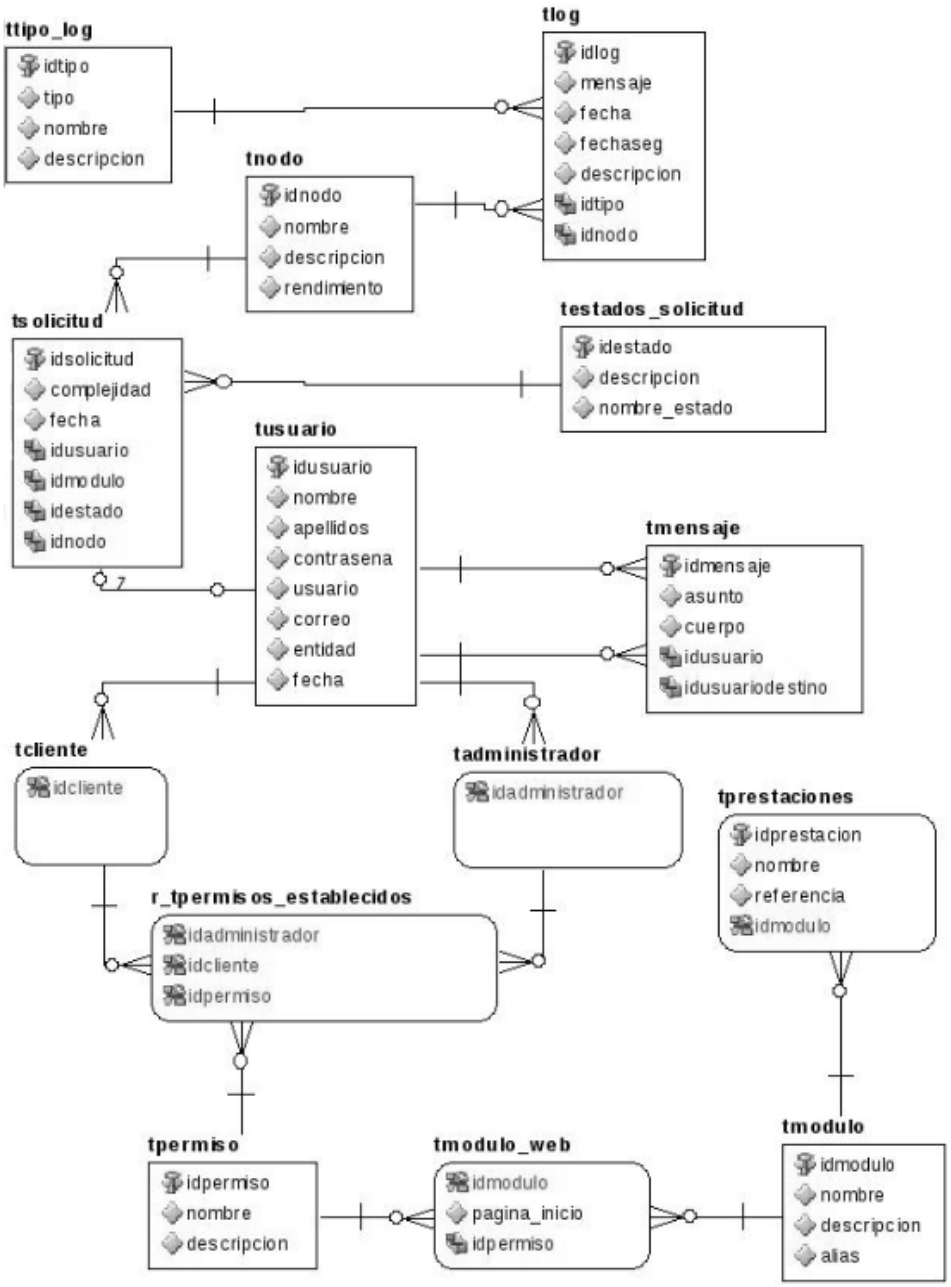

Figura 3.4: Submodelo Entidad Relación para la Administración y Monitorización de la plataforma.

Descripción:

El submodelo de datos anterior corresponde a la siguiente descripción. Cada módulo brinda un conjunto de prestaciones y una prestación pertenece a un único módulo. Los módulos pueden ser módulos de cálculo o módulos de aplicación (denominados módulos web). A cada módulo de cálculo le corresponde un módulo de aplicación que es con el que interactúa directamente el usuario para ejecutar el módulo de cálculo correspondiente, sin embargo existen módulos de aplicación que no corresponden a ningún módulo de cálculo, por ejemplo los módulos de Administración de Usuarios y Gestión de Errores.

Cada módulo de aplicación está asociado a un permiso. Los permisos son asignados por los administradores a los clientes, esta información se almacena en la tabla r_tpermisos_establecidos.

En la plataforma los usuarios pueden compartir los resultados que obtienen a partir de las solicitudes que ejecutan. Estos resultados se envían en forma de mensaje. Un usuario puede enviar y recibir muchos mensajes independientemente del rol que tengan en el sistema. La información de estos mensajes se almacenan en la tabla tmensaje.

Los usuarios pueden ser clientes o administradores. Aunque los clientes y los administradores no tengan información adicional de los usuarios, esta generalización/especialización de tusuario con tcliente y tadministrador se diseña utilizando tablas independientes para cada relación producto a que la plataforma esta en una fase inicial de desarrollo y en un futuro si pueden incrementarse los atributos de estas entidades.

Cada solicitud es realizada por un usuario y cada usuario puede realizar muchas solicitudes siempre que estas pertenezcan a los módulos de cálculo a los que este tenga permiso.

De cada nodo se puede consultar el reporte de registros que este ha arrojado.

3.4.2. Descripción de las tablas.

En esta sección se describen de forma general los datos que se almacenan en cada tabla del modelo de datos de la Base de Datos de la plataforma Génesis y se explican brevemente lo que representan todos sus atributos.

Nombre de la tabla: tsolicitud		
Descripción: Almacena la información de las solicitudes de cálculo que se realizan.		
Atributo	**Tipo**	**Descripción**
idsolicitud	bigint	Identificador de la solicitud.
fecha	date	Fecha en que se realiza la solicitud
complejidad	integer	Valor asociado a la complejidad de la solicitud. Dependiente del volumen de información que procesa la solicitud.
idusuario	integer	Identificador del usuario que realiza la solicitud.
idnodo	integer	Identificador del nodo en el que se ejecuta la subtarea máster de la solicitud.
idmodulo	integer	Identificador asociado al módulo de cálculo al que pertenece la solicitud.
idestado	integer	Identificador asociado al estado en que se encuentra la solicitud.

Cuadro 3.8: Tabla tsolicitud

Nombre de la tabla: testados_solicitud		
Descripción: Almacena los tipos de estados que pueden tener las solicitudes en dependencia del nivel de procesamiento en que se encuentran.		
Atributo	**Tipo**	**Descripción**
idestado	bigint	Identificador único de la tabla testados_solicitud.
descripción	varchar(80)	Descripción detallada del estado de la solicitud.
nombre_estado	varchar(20)	Nombre del estado de la solicitud.

Cuadro 3.9: Tabla testados_solicitud

Nombre de la tabla: tprestaciones		
Descripción: Almacena las prestaciones que tiene un módulo de cómputo. Representan las funcionalidades de un módulo acoplado a la plataforma.		
Atributo	**Tipo**	**Descripción**
idprestacion	integer	Identificador único de la tabla tprestaciones.
nombre	varchar(80)	Descripción detallada del estado de procesamiento de una solicitud
referencia	varchar(80)	Referencia a la página que carga la prestación.
idmodulo	integer	Identificador del módulo al que pertenece la prestación.

Cuadro 3.10: Tabla tprestaciones

Nombre de la tabla: tadministrador		
Descripción: Almacena la información relacionada con los administradores del sistema.		
Atributo	**Tipo**	**Descripción**
idadministrador	integer	Identificador que representa los usuarios del sistema que son administradores.

Cuadro 3.11: Tabla tadministrador

Nombre de la tabla: tcliente		
Descripción: Almacena la información relacionada con los usuarios que son clientes del sistema.		
Atributo	**Tipo**	**Descripción**
idprestacion	integer	Identificador único de la tabla tcliente

Cuadro 3.12: Tabla tcliente

Nombre de la tabla: tcluster		
Descripción: Almacena la información de los clúster de procesamiento que manipula la plataforma. La plataforma es flexible para la manipulación de varios clúster de procesamiento.		
Atributo	**Tipo**	**Descripción**
idcluster	integer	Identificador único de la tabla tprestaciones.
nombre	varchar(80)	Descripción detallada del estado de procesamiento de una solicitud
descripción	varchar(80)	Referencia a la página que carga la prestación.
idservidor	integer	Identificador del nodo que funciona como servidor para esa configuración.

Cuadro 3.13: Tabla tcluster

Nombre de la tabla: tlog		
Descripción: Almacena la información de los registros que arrojan los nodos del clúster de procesamiento.		
Atributo	**Tipo**	**Descripción**
idlog	bigint	Identificador único de la tabla tlog.
mensaje	varchar(255)	Mensaje asociado al registro.
fecha	date	Fecha en que se emitió.
fechaseg	integer	Fecha en segundos.
descripcion	varchar(255)	Descripción detallada del registro.
idtipo	integer	Tipo de registro emitido.
idnodo	integer	Nodo que lo emitió.

Cuadro 3.14: Tabla tlog

Nombre de la tabla: tmensaje		
Descripción: Almacena la información de los mensajes que se envían los usuarios para compartir información de los resultados de las solicitudes que procesan.		
Atributo	**Tipo**	**Descripción**
idmensaje	bigint	Identificador único de la tabla tmensaje.
asunto	varchar(100)	Asunto del mensaje.
cuerpo	varchar(255)	Cuerpo del mensaje.
idusuario	integer	Identificador del usuario que envía el mensaje.
idusuariodestino	integer	Identificador del usuario que recibe el mensaje.

Cuadro 3.15: Tabla tmensaje

Nombre de la tabla: tmodulo		
Descripción: Almacena la información de los módulos generales que se acoplan a la plataforma. Pueden ser módulos de cálculo y/o módulos de aplicación.		
Atributo	**Tipo**	**Descripción**
idmodulo	integer	Identificador único de los módulos del sistema.
descripción	varchar (255)	Descripción de los diferentes módulos.
nombre	varchar (20)	Nombre de los módulos de la plataforma.
alias	varchar (4)	Alias asociado al nombre del módulo de cómputo.

Cuadro 3.16: Tabla tmodulo

Nombre de la tabla: tmodulo_calculo		
Descripción: Almacena la información de los módulos de cálculo. Los módulos de cálculo contienen la información necesaria para el cálculo distribuido de los módulos incorporados a la plataforma.		
Atributo	**Tipo**	**Descripción**
idmodulo	integer	Identificador único de la tabla tmodulo.

Cuadro 3.17: Tabla tmodulo_calculo

Nombre de la tabla: tmodulo_web		
Descripción: Almacena la información de los módulos de aplicación.		
Atributo	**Tipo**	**Descripción**
idmodulo	integer	Identificador único de la tabla tmodulo.
pagina_inicio	varchar(80)	Identificador de la página relacionada con el módulo.
idpermiso	integer	Identificador del permiso asociado a ese módulo.

Cuadro 3.18: Tabla tmodulo_web

Nombre de la tabla: tnodo_esclavo		
Descripción: Almacena la información de los nodos del clúster de procesamiento que son esclavos.		
Atributo	**Tipo**	**Descripción**
idesclavo	integer	Identificador de la tabla tnodo_esclavo que representa los nodos que son esclavos.
idcluster	integer	Identificador del clúster al que pertenece dicho nodo esclavo.

Cuadro 3.19: Tabla tnodo_esclavo

Nombre de la tabla: tnodo_servidor		
Descripción: Almacena la información de los nodos del clúster de procesamiento que son servidores.		
Atributo	**Tipo**	**Descripción**
idservidor	integer	Identificador único de la tabla tnodo_servidor que representa a los nodos que son servidores.

Cuadro 3.20: Tabla tnodo_servidor

Nombre de la tabla: tusuario		
Descripción: Almacena la información de todos los usuarios del sistema. Estos pueden ser Administradores o Clientes.		
Atributo	**Tipo**	**Descripción**
idusuario	integer	Identificador único de la tabla tusuario.
nombre	varchar(30)	Nombre del usuario.
apellidos	varchar(60)	Apellidos del usuario.
contrasena	varchar(25)	Contraseña del usuario para la cuenta en el sistema.
usuario	varchar(30)	Nombre de la cuenta del usuario.
correo	varchar(60)	Cuenta de correo del usuario.
entidad	varchar(15)	Nombre de la entidad a la que pertenece el usuario.
fecha	date	Fecha de inscripción del usuario al sistema.

Cuadro 3.21: Tabla tusuario

Nombre de la tabla: tpagina		
Descripción: Almacena la información del paginado por cada nodo.		
Atributo	**Tipo**	**Descripción**
idpagina	bigint	Identificador único de la tabla tpagina.
Id_subtarea_remota	integer	Subtarea remota que recibe el mensaje.
Id_subtarea_local	integer	Subtarea local que envía el mensaje.
tipo_dato	integer	Tipo de dato asociado al mensaje.
dato	varchar(120)	Dato que se envía.
id_etiqueta	integer	Etiqueta con que se envía el mensaje.
idnodo	integer	Identificador del nodo que realiza el paginado.
Iduc8	bigint	Identificador del mensaje.

Cuadro 3.22: Tabla tpagina

Nombre de la tabla: ttipo_log		
Descripción: Almacena los tipos de registro de información del sistema.		
Atributo	**Tipo**	**Descripción**
idtipo	integer	Identificador único de la tabla ttipo_log.
tipo	varchar(30)	Tipo de registro.
descripcion	varchar(80)	Descripción del registro.
nombre	varchar(40)	Nombre del registro.

Cuadro 3.23: Tabla ttipo_log

Nombre de la tabla: tpermiso		
Descripción: Almacena los permisos que se le conceden a los usuarios del sistema.		
Atributo	**Tipo**	**Descripción**
idpermiso	integer	Identificador único de la tabla tpermiso.
descripcion	varchar(80)	Descripción detallada del permiso.
nombre	varchar(20)	Nombre del permiso que se concede.

Cuadro 3.24: Tabla tpermiso

Nombre de la tabla: r_tpermisos_establecidos		
Descripción: Almacena los permisos que concede un administrador a un cliente.		
Atributo	**Tipo**	**Descripción**
idcliente	integer	Identificador del usuario cliente que recibe el permiso.
idadministrador	integer	Identificador del usuario administrador que otorga el permiso.
idpermiso	integer	Identificador del permiso otorgado.

Cuadro 3.25: Tabla r_tpermisos_establecidos

Nombre de la tabla: tnodo		
Descripción: Almacena la información de los nodos de un clúster de procesamiento.		
Atributo	**Tipo**	**Descripción**
idnodo	integer	Identificador del nodo.
nombre	varchar(20)	Nombre del nodo.
descripcion	varchar(100)	Información adicional del nodo.
rendimiento	double precision	Valor asociado a la capacidad de procesamiento del nodo.

Cuadro 3.26: Tabla tnodo

3.5. Descripción de componentes que tienen acceso a Bases de Datos.

Explorador de Solicitudes.

El Explorador de Solicitudes es un componente de la plataforma que interactúa directamente con la Base de Datos para manipular las solicitudes de cómputo que realizan los usuarios. No es objetivo explicar la interacción del componente en la plataforma, sino la manera en que interactúa con las clases persistentes para la implementación de sus funcionalidades.

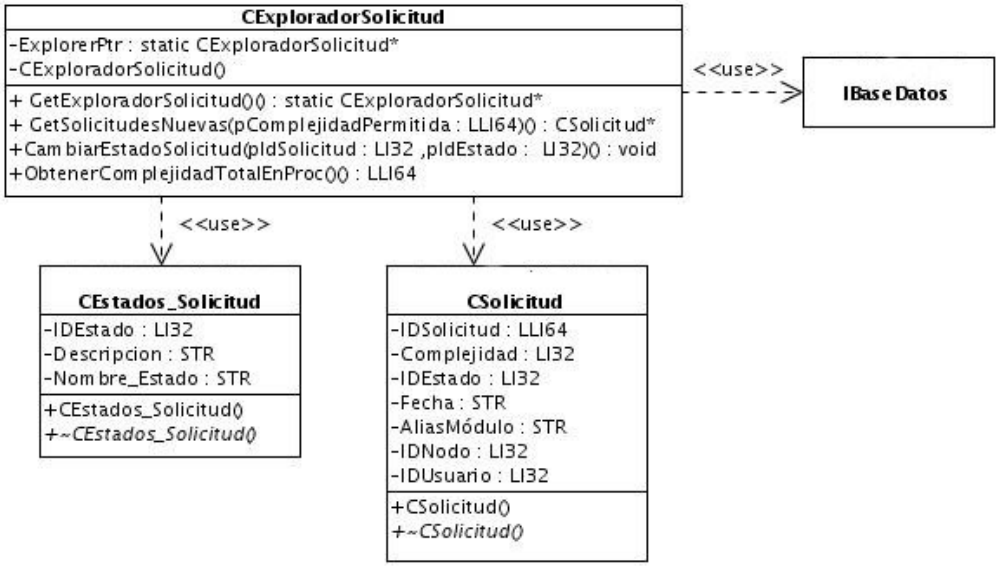

Figura 3.5: Diagrama de Clases Explorador de Solicitudes.

El Explorador de Solicitudes se relaciona con las clases persistentes CEstados_Solicitud y CSolicitud para realizar operaciones con sus tablas correspondientes mediante las funcionalidades *void CambiarEstadoSolicitud(LI32 pIdSolicitud, LI32 pIdEstado)* y *CSolicitud* GetSolicitudesNuevas (LLI64 pComplejidadPermitida)*. En el caso de las funcionalidades *CSolicitud* GetSolicitudesNuevas(LLI64 pComplejidadPermitida)* y *LLI64 ObtenerComplejidadTotalEnProc()*, las consultas se realizan a través de la clase abstracta IBaseDatos debido a su complejidad. Se ofrece la descripción de la clase CExploradorSolicitud para una mayor comprensión.

Nombre de la clase: CExploradorSolicitud	
Descripción: Clase controladora que ofrece las funcionalidades para manipular las solicitudes de cómputo.	
Atributos	**Tipo**
ExplorerPtr	static CExploradorSolicitud*
Responsabilidades de la clase:	
Nombre:	CCExploradorSolicitudes()
Descripción:	Constructor de la clase sin parámetros.
Nombre:	~CExploradorSolicitudes()
Descripción:	Destructor de la clase.
Nombre:	static CExploradorSolicitud* GetExploradorSolicitud()
Descripción:	Devuelve la instancia de la clase CExploradorSolicitud.
Nombre:	CSolicitud* GetSolicitudesNuevas(LLI64 pComplejidadPermitida)
Descripción:	Obtiene una solicitud de la base de datos cuya complejidad es menor que la que se especifica por parámetro.
Nombre:	void CambiarEstadoSolicitud(LI32 pIdSolicitud, LI32 pIdEstado)
Descripción:	Cambia a partir del id de la solicitud su estado.
Nombre:	LLI64 ObtenerComplejidadTotalEnProc()
Descripción:	Devuelve la suma total de la complejidad de cada una de las solicitudes que se están procesando.

Cuadro 3.27: Descripción de la Clase CExploradorSolicitudes

Notificador de Registros

El Notificador de Registros es el componente que tiene la plataforma para la notificación de reportes de sucesos del sistema. Cada nodo realiza su propio reporte de registros.

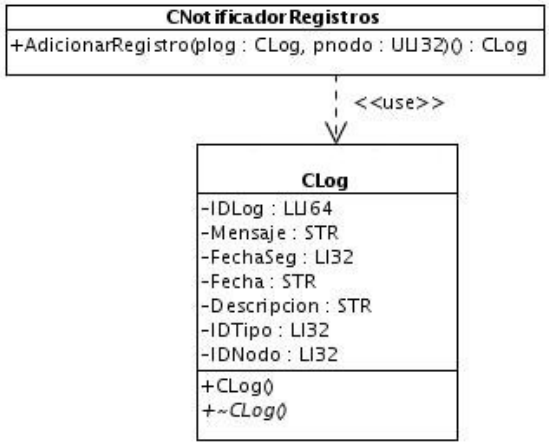

Figura 3.6: Diagrama de clases del Notificador de registros.

Nombre de la clase: CNotificadorRegistro	
Descripción: Clase controladora para la notificación de registros.	
Responsabilidades de la clase:	
Nombre:	CLog AdicionarRegistro(CLog pLog, ULI32 pNodo)
Descripción:	Método que permite adicionar un nuevo registro a la Base de datos

Cuadro 3.28: Descripción de la Clase CNotificadorRegistro

La única funcionalidad asociada a este componente es la inserción de un registro en la Base de Datos de ahí que se relacione nada más con la clase persistente CLog.

Paginado.

El Paginado es uno de los componentes que más interactúa con la Base de datos. Utilizado por el Buffer de Mensajes para insertar en la Base de Datos los mensajes recibidos cuando su capacidad de almacenamiento ha llegado al límite.

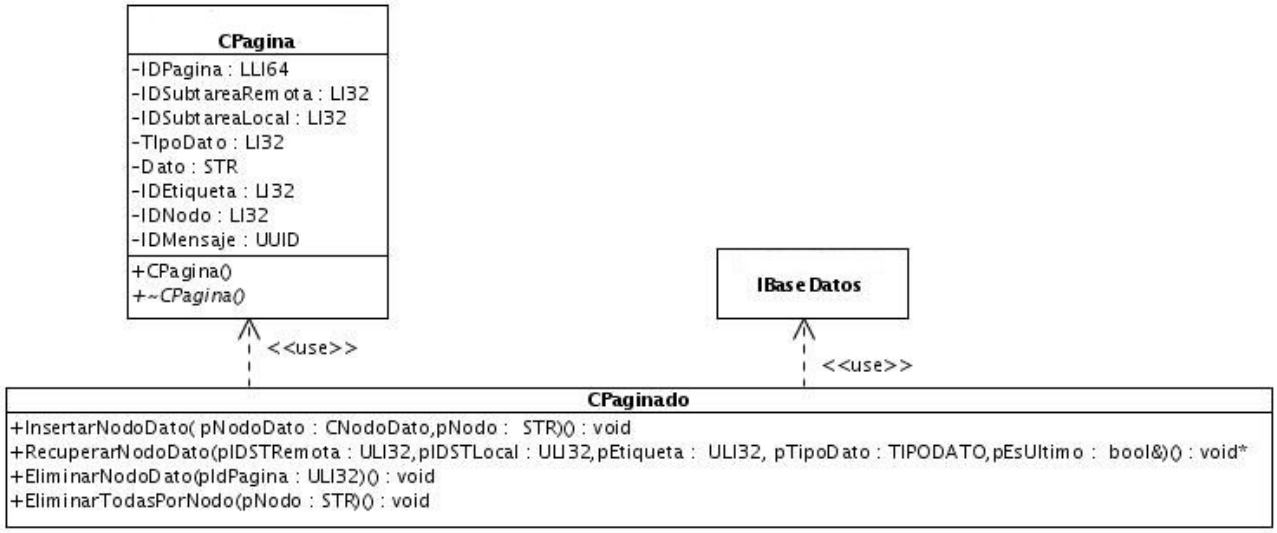

Figura 3.7: Diagrama de clases del Paginado.

Nombre de la clase: CPaginado	
Descripción: Clase controladora que utiliza el Buffer de Mensajes para gestionar los mensajes que pagina a la Base de Datos.	
Responsabilidades de la clase:	
Nombre:	void InsertarNodoDato()
Descripción:	Método que permite insertar un nuevo nodo.
Nombre:	void* RecuperarNodoDato(ULI32 pIDSTRemota,ULI32 pIDSTLocal, ULI32 pEtiqueta, TIPODATO pTipoDato, bool &pEsUltimo)
Descripción:	Devuelve el dato asociado a los parámetros señalados.
Nombre:	void EliminarNodoDato(ULI32 pIdPagina)
Descripción:	Método que permite eliminar un nodo dado el id.
Nombre:	void EliminarTodasPorNodo(STR pNodo)
Descripción:	Elimina todo el paginado de un nodo dado su nombre.

Cuadro 3.29: Descripción de la Clase CPaginado

En este caso interactúa con la clase CPagina para operaciones sobre su tabla correspondiente en la implementación de las funciones *void InsertarNodoDato()* y *void EliminarNodoDato(ULI32 pIdPagina)*. Utiliza la clase abstracta IBaseDatos para realizar consultas complejas en la implementación de las funciones *void* RecuperarNodoDato(ULI32 pIDSTRemota,ULI32 pIDSTLocal, ULI32 pEtiqueta, TIPODATO pTipoDato, bool &pEsUltimo)* y *void EliminarTodasPorNodo(STR pNodo)*.

3.6. Especificaciones complementarias del acceso a datos para la incorporación de módulos de cómputo a la plataforma Génesis.

Los módulos de cómputo que se incorporan a la plataforma tienen asociado un modelo de datos propio que reside en una o varias Bases de Datos externas a la base de datos de Génesis. Para adicionar un módulo de cómputo a la plataforma es necesario en ambos casos seguir las siguientes especificaciones, referidas al acceso a datos:

1. **Configuración de los parámetros de las Bases de Datos.**

   Cuando los módulos de cálculo acceden a Bases de Datos externas, es necesario especificar los parámetros de configuración para la conexión a dichas Bases de Datos. En este caso se deben seguir la descripción en el *epígrafe 2.3 (3)* de este documento.

2. **Diseño de la base de datos**

   Para el diseño del modelo de datos de los módulos es aconsejable seguir las normas de diseño de Bases de Datos que propone la plataforma. El diseño del modelo de datos influye en la optimización de las consultas, en el Documento de Estándares de Diseño de Bases de Datos de la Plataforma Génesis [1] se proponen algunas técnicas utilizadas que pueden ser aplicadas para un mejor desempeño del módulo de cómputo que se desarrolla. La aplicación de estos estándares permiten además mayor entendimiento del diseño, aunque la utilización o no de estos no afectará el funcionamiento de la plataforma puesto que está construida de manera flexible para consultar Bases de Datos que han sido creadas ya previamente con otras normas de diseño.

3. **Diseño de Clases Persistentes.**

   El diseño de clases persistentes se realiza a partir del modelo de datos, puesto que el Motor de Persistencia de la plataforma no genera el código SQL asociado a la creación de objetos en la base de datos.

   Las clases persistentes se crean como se explica en el *epígrafe 2.3 (1)*. Una vez diseñadas las clases a partir del modelo relacional, se propone crear una clase que encapsule el comportamiento de las operaciones que se realizan con la base de datos de un determinado componente denominada

[1] El Documento de Estándares de Diseño de Bases de Datos de la Plataforma Génesis se muestra en el Anexo #1.

Controladora de Datos. Es una clase, generalmente única para módulos de cómputo de cierta complejidad, que contiene la implementación de las funciones que se realizan en el módulo referidas al acceso a datos, de manera que puedan ser llamadas desde otras partes del módulo. Las clases *CExploradorSolicitud*, *CPaginado* y *CNotificadorErrores* son clases Controladoras de Datos que sirven al MPPP.

La Controladora de Datos de cada módulo interactúa con las clases persistentes que este genera y las clases herederas de IBaseDatos que repesentan las Bases de Datos a las que puede conectarse para la implementación de sus funcionalidades.

4. **Optimización de Consultas.**

Una de las características que tiene la plataforma Génesis que la distinguen del resto de los software similares son las facilidades que ofrece para el trabajo con Bases de Datos. Todos los módulos de cálculo que pueden acoplarse a la plataforma, en su mayoría realizan transacciones complejas en Bases de Datos. Estas transacciones son elaboradas por el programador en lenguaje SQL y la forma de elaborar las consultas puede mejorar el rendimiento del módulo de cómputo.

Algunos de los aspectos que influyen en la optimización de las consultas SQL son:

a) **Campos en las consultas de recuperación de datos:**

1) Cuando se realizan consultas que recuperan campos de varias tablas deben especificarse el nombre de las tablas y así el SGBD no emplea tiempo en buscar en cuál de las tablas se localiza el atributo.
   Ejemplo: En lugar de emplear la consulta "SELECT idnodo, nombre, descripcion, rendimiento, idcluster FROM tnodo, tnodo_esclavo WHERE idnodo = idesclavo", para obtener los datos de todos los nodos que han trabajado en modo esclavo debería emplearse "SELECT tnodo.idnodo, tnodo.nombre, tnodo.descripcion, tnodo.rendimiento, tnodo_esclavo.idcluster FROM tnodo_esclavo, tnodo WHERE tnodo_esclavo.idesclavo = tnodo.idnodo".
2) Se deben seleccionar siempre los campos que se necesiten únicamente, así se optimiza tiempo y además espacio.
3) Cuando se seleccionan todos los campos de una relación, deben especificarse los nombres de los campos en lugar de poner el símbolo (*), así se evita el SGBD tenga que leer primero la estructura de la tabla antes de ejecutar la sentencia.

Ejemplo: En lugar de emplear la consulta "SELECT * FROM tpagina" debería emplearse "SELECT tpagina.idpagina, tpagina.idsubtarea_local, tpagina.idsubtarea_remota, tpagina.tipodato, tpagina.dato, tpagina.idetiqueta, tpagina.idnodo, tpagina.idmensaje_uc8 FROM tpagina".

b) **Elección de los índices:** Los índices deben elegirse con cuidado, solo se recomiendan para atributos que sirven de nexo entre tablas que se consultan constantemente y no sobre campos triviales que no se utilicen en el proceso de búsqueda, debido a que introducen un problema de sobrecarga de memoria y ralentiza otras tareas en la base de datos.

c) **Orden de las Tablas en la cláusula FROM:** Cuando se necesite recuperar campos de varias tablas, el orden de las tablas después de la cláusula "FROM" es determinante. Deben ordenarse por la cantidad de registros que puedan tener.

Ejemplo: La tabla tsolicitud es propensa a tener una cantidad de registros considerablemente mayor que la tabla testados_solicitud por lo que en lugar de "SELECT tsolicitud.idsolicitud, testado_solicitud.nombre_estado FROM tsolicitud, testados_solicitud WHERE tsolicitud.idestado=testados_solicitud.idestado AND testados_solicitud.idestado = 2" donde recorrería todos los registros de la tabla tsolicitud, debería ejecutarse la consulta "SELECT tsolicitud.idsolicitud, testado_solicitud.nombre_ estado FROM testados_solicitud, tsolicitud WHERE testados_solicitud.idestado = 2 AND testados_solicitud.idestado = tsolicitud.idestado" donde primero se filtran los estados y después se seleccionan las solicitudes, así se recorren menos registros.

d) **Unión de tablas:** Siempre que se pueda, debe utilizarse INNER JOIN , LEFT JOIN o RIGHT JOIN para unir las tablas en lugar del WHERE, esto permite que a medida que se declaran las tablas se van uniendo mientras que si se utiliza el WHERE el SGBD genera primero el producto cartesiano de todos los registros de las tablas para luego filtrar las correctas, lo que representa un trabajo definitivamente lento.

Ejemplo: Para seleccionar las solicitudes y el nombre del estado en que se encuentran, en lugar de ejecutar "SELECT testado_solicitud.nombre_estado, tsolicitud.idsolicitud FROM testados_solicitud, tsolicitud WHERE testados_solicitud.idestado = tsolicitud.idestado AND testados_solicitud.idestado = 2" debería ejecutarse "SELECT testados_solicitud.nombre_estado, tsolicitud.idsolicitud FROM testados_solicitud INNER JOIN tsolicitud ON (testados_solicitud.idestado = tsolicitud.idestado) WHERE testado_solicitud.idestado = 2"

El proceso de optimización de consultas es bastante extenso, solo se exponen algunos elementos simples que pueden influir en la eficiencia de los módulos de cómputo cuando la cantidad de transacciones que ejecutan es muy grande.

3.7. Conclusiones.

La Capa de Acceso a Datos de la plataforma Génesis media entre los Motores de Procesamiento de Peticiones Paralelas y la Base de Datos. Implementada con el Motor de Persistencia de la plataforma trata de satisfacer al máximo sus requerimientos.

Capítulo 4

Disponibilidad de Datos.

4.1. Introducción

La disponibilidad del servicio de acceso a datos es otra de las premisas que tiene Génesis. En este capítulo se describe el Modelo de Despliegue de la plataforma, y la alternativa propuesta para garantizar la disponibilidad y el rendimiento del SGBD validada por un Caso de Pruebas de Estrés.

4.2. Modelo de Despliegue de la plataforma Génesis.

En la *figura 4.1* se presenta una propuesta de distribución de los nodos físicos del clúster de Bases de Datos para la plataforma Génesis. Este modelo se adapta a las características de la *figura 1.2,* además se le añade el Servidor de Balance de Carga Alternativo para garantizar mayor disponibilidad del servicio utilizando la herramienta Heartbeat.

El clúster de Bases de Datos tiene dos Servidores de Bases de Datos que son actualizados por un Servidor de Réplica. En la configuración se propone esta proporción para el modo de funcionamiento más estable de Génesis con el cuál se ha estado desarrollando hasta la actualidad. A medida que aumentan las transacciones al clúster de Bases de Datos realizadas por el clúster de procesamiento de la plataforma, puede incrementarse la cantidad de nodos de este tipo obteniendo mejores resultados. Sin embargo la determinación de un cambio en el modelo tendría que estar respaldada por un proceso extensivo de pruebas, aunque el modelo es escalable, el incremento de los Servidores de Bases de Datos provoca aumento del tiempo del proceso de réplica.

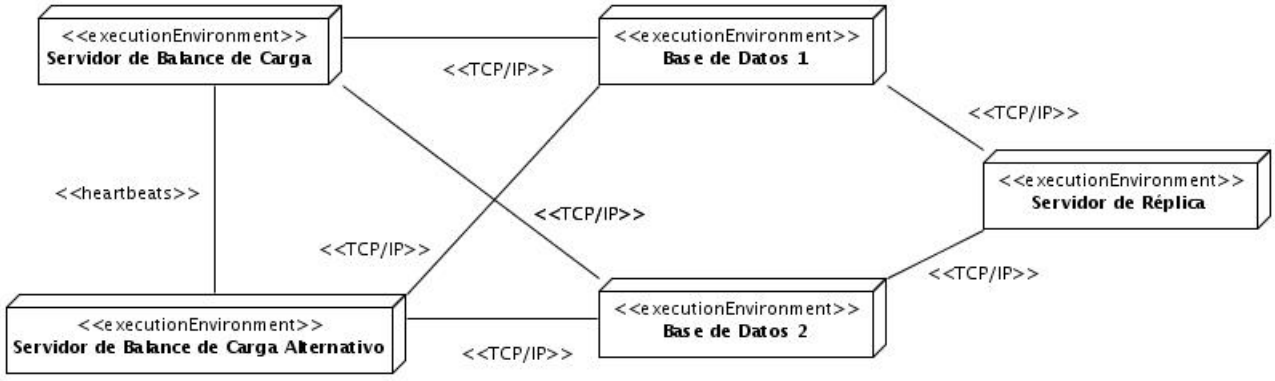

Figura 4.1: Modelo de Despliegue del Clúster de Bases de Datos

En la *figura 1.2* se muestra el Balanceador de Carga como el nodo que inicia el flujo de eventos recibiendo las peticiones de los nodos del clúster de procesamiento de Génesis. El Balanceador de Carga es el nodo que ocasiona mayor daño en caso de que se detengan sus servicios debido al papel que tiene en el clúster de Bases de Datos. El cese de sus funciones determina la pérdida completa de la conexión a la base de datos. Un fallo en la conexión genera trabajo con sobrecarga, no completamiento de las transacciones y errores en el funcionamiento de la aplicación. Es por eso que se identifica como un factor de riesgo notable y garantizar la disponibilidad de sus servicios es una tarea de gran prioridad. Ubicar un servidor alternativo que asuma los servicios de este en caso de un fallo en su funcionamiento, aumentará el nivel de disponibilidad de acceso a los datos. En la *figura 4.2* se muestra cómo el servidor alternativo asume los servicios del Servidor de Balance de Carga cuando ocurre un error, el margen de tiempo de restablecimiento es configurable.

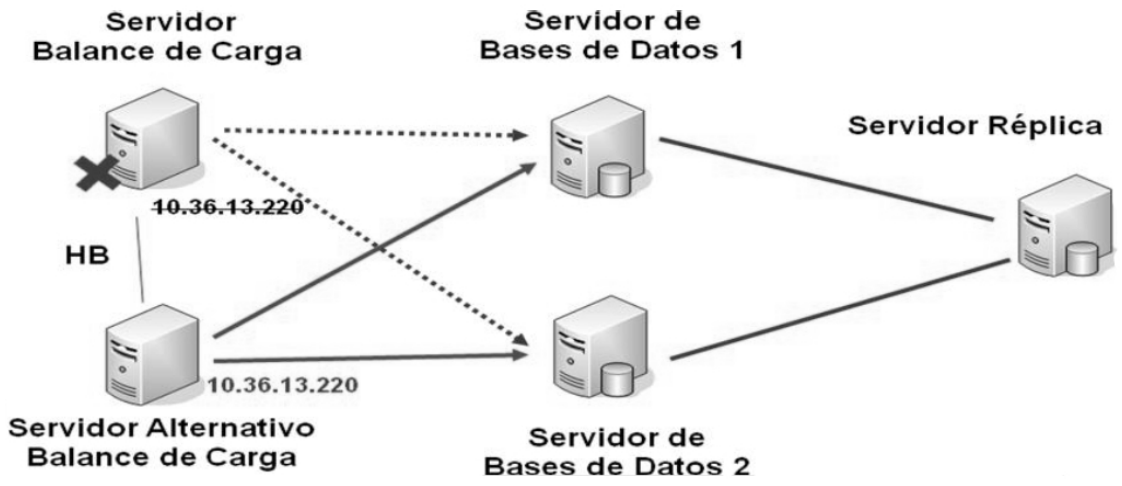

Figura 4.2: Recuperación del servicio ante el fallo del Servidor de Balance de Carga.

Aunque la solución que nos ofrece Heartbeat es muy factible, no es aconsejable aplicarla para el resto de los servidores porque:

1. La herramienta PgCluster maneja hasta ciertos niveles la tolerancia a fallos, lo que nos garantiza robustez en la solución planteada.

   Si existe un fallo en un nodo de base de datos (Servidor de Bases de Datos) los nodos homólogos a este pueden continuar el servicio asumiendo la próxima carga de tareas, el Balanceador de Carga lo elimina como posible receptor de consultas y los Servidores de Réplica no lo actualizan hasta tanto no se active nuevamente. En la *figura 4.3* se muestra este escenario.

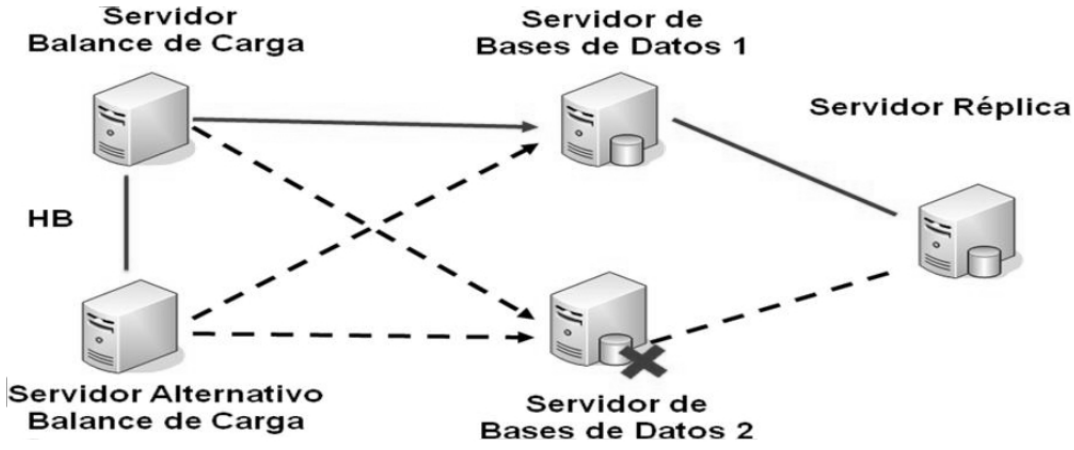

Figura 4.3: Tolerancia ante fallos en un Servidor de Bases de Datos.

Si falla uno de los Servidores de Réplica, los Servidores de Bases de Datos asociados a este funcionan en modo "solo lectura" [1], el Balanceador de Carga no dirige ninguna consulta de escritura hacia ellos, solo de lecturas y el resto de los Servidores de Réplica continúan actualizando sus datos para que puedan seguir siendo consultados. Existen Servidores de Réplica que se configuran para asumir el servicio de otro en caso que falle, pero esta técnica no se utiliza en el modelo presentado. En el caso que fallen todos los replicadores, el clúster de Bases de Datos funcionará completamente en modo "solo lectura" como se muestra para la configuración del clúster de Génesis en la *figura 4.4.*

[1] El modo "solo lectura" es referido en la bibliografía frecuentemente como "stand-alone" por sus siglas en Inglés.

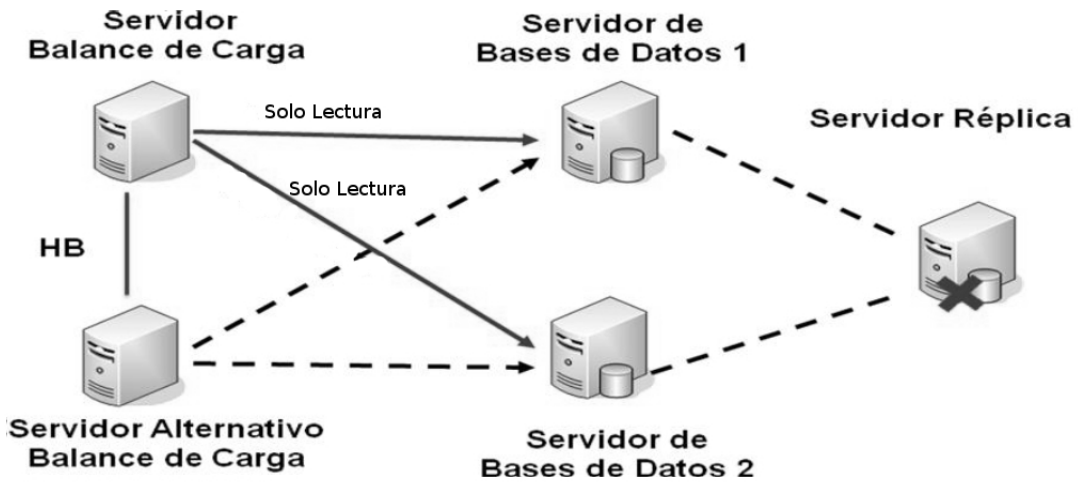

Figura 4.4: Tolerancia ante fallos en un Servidor de Réplica.

2. Incrementar el número de servidores alternativos aumenta considerablemente el tráfico de paquetes en la red, para las actividades de monitoreo a los servidores originales. En el caso de las Redes de Área Local puede ocasionar retraso en el flujo normal de información y saturación de la red.

3. Inversión innecesaria en recursos adicionales para aplicaciones pequeñas.

En el Anexo 3 y Anexo 4 se muestran las configuraciones para las herramientas PgCluster y HeartBeat respectivamente que se utiliza en la plataforma.

4.3. Caso de Prueba de Estrés a la Disponibilidad de Datos.

Este Caso de Prueba está basado en la ejecución del módulo de cómputo Megamatrices. Dirigido a evaluar la variación del rendimiento del procesamiento de una misma petición de cómputo con la distribución de los datos almacenados.

Descripción General.

Se realizan dos solicitudes simultáneas del módulo Megamatrices, actualmente incorporado a la plataforma que permite la multiplicación de matrices de forma paralela. Por cada solicitud que se

procesa se crea una subtarea máster que envía a cada subtarea esclava el número de la fila (numero_fila) que le corresponde procesar de la primera matriz, su identificador (idmatriz_entrada_uno) y su cantidad de columnas (cantidad_columnas_me_uno), el identificador de la segunda matriz (idmatriz_entrada_dos), su cantidad de columnas (cantidad_columnas_me_dos) y el identificador de la matriz resultado (idmatriz_resultado).

Cada subtarea de cálculo extrae de la base de datos el vector fila que el corresponde procesar de la tabla tvalores_matriz y lo multiplica por cada columna de la matriz de entrada dos, los valores de cada columna de la matriz de entrada dos se extraen individualmente. Los resultados parciales que se van obteniendo se insertan en la tabla tvalores_matriz a partir del identificador 'idmatriz_resultado'.

Esta prueba no está destinada a medir la eficiencia del algoritmo de multiplicación de matrices pues la implementación utilizada no es la más eficiente para resolverlo, además la matriz utilizada es de tamaño pequeño para obtener resultados de prueba en poco tiempo. Tiene como objetivo realizar la mayor cantidad de consultas posibles en una solicitud de cálculo sin tener en cuenta su costo, para demostrar el aumento del rendimiento de un algoritmo paralelo que realice una carga de consultas grande, cuando mejora la distribución de la base de datos.

Condiciones de Ejecución.

Este caso de prueba se ejecutó en un clúster de Red de Estaciones de Trabajo (NOW, por sus siglas en inglés) con cuatro computadoras (3 nodos esclavos (MPPPC) , 1 nodo servidor de aplicación (MPPPS)), de dos núcleos Intel(R) Pentium(R) CPU 3.00GHz, Memoria RAM 512 MiB con plataforma Linux, distribución Debian/GNU 2.6.26-2-686 conectadas por una red Ethernet de 100 Mb/s.

Escenarios de Pruebas de Estrés.

<table>
<tr><th>Carga de Trabajo</th><th>Escenario</th><th>Variante</th></tr>
<tr><td rowspan="2">Cantidad de solicitudes a procesar: 2
Cantidad de subtareas máster: 42
Orden Matriz Entrada 1: 20 x 100
(20 filas x 100 columnas)
Orden Matriz Entrada 2: 100 x 200
(100 filas x 200 columnas)</td><td>Escenario #1</td><td>Datos almacenados
en una base de datos</td></tr>
<tr><td>Escenario #2</td><td>Datos almacenados en un
SGBD distribuido.</td></tr>
</table>

Descripción Escenario #1

El SGBD de la plataforma se encuentra en uno de los nodos del clúster (eipad5) que funciona además como nodo servidor del Clúster de Procesamiento. El resto de los nodos esclavos (eipad2, eipad3 y eipad4) realizan las consultas para la ejecución del algoritmo. Se realizan para el procesamiento un total de 808002 consultas a la BD, 8000 consultas de escritura y 800002 consultas de lectura.

Resultados de la prueba Escenario #1

Tiempo de Duración: 09:10:00 am - 10:09:54 am. 59 min 54 seg.[2]

Descripción Escenario #2

El SGBD de la plataforma se encuentra distribuido en los nodos del clúster, dos ClusterBD (eipad2 y eipad3) y un Replicador (eipad4), que funcionan además como nodos esclavos del Clúster de Procesamiento. El nodo servidor (MPPPS) está localizado en eipad5 y funciona además como nodo de balance de carga. Los nodos realizan un total de 808 002 consultas a la BD, 8000 consultas de escritura y 800 002 consultas de lectura.

Resultados de la prueba Escenario #2

Tiempo de Duración: 10:39:00 am - 11:11:10 am. 32 min 10 seg.[3]

Conclusiones del Caso de Prueba.

Para la ejecución de dos solicitudes de computo del mismo algoritmo sobre la plataforma Génesis se realizaron 404001 consultas por cada solicitud. A partir de la misma carga se generaron dos escenarios cuya diferencia estuvo marcada en la distribución de los datos. Los resultados de la prueba arrojaron la reducción casi a la mitad del tiempo de ejecución cuando el SGBD se encuentra distribuido. Por tal motivo distribuir los datos en la plataforma nos permite ganar en eficiencia en el procesamiento de las solicitudes.

[2]Calculado por funciones de la librería «time» de C++.
[3]Calculado por funciones de la librería «time» de C++.

4.4. Conclusiones

Aunque los esfuerzos sean considerables para lograr la alta disponibilidad de los sistemas, es muy difícil alcanzar una completa tolerancia a fallos. Los grupos de desarrollo se concentran en cubrir gradualmente los riesgos que causan mayores afectaciones. En este capítulo se presentó la alternativa diseñada para la disponibilidad del servicio de acceso a datos en la plataforma de cómputo paralelo Génesis que se ha estado utilizando hasta el momento. La ejecución de un caso de prueba demuestra la efectividad de la distribución de los datos que se revierte en un mejor rendimiento del procesamiento de las solicitudes de cómputo que se ejecutan.

Conclusiones

Al concluir el presente trabajo de diploma podemos afirmar que se realizó un análisis de los principios para lograr mejoras en la persistencia, acceso y disponibilidad en sistemas que procesan grandes volúmenes de datos. Se cumplieron los objetivos propuestos a partir de los siguientes resultados alcanzados:

- Se logró el diseño y la implementación de un Motor de Persistencia acoplado a la plataforma Génesis que facilita el acceso a Bases de Datos para las aplicaciones multihilos desarrollada en C++ solucionando las diferencias entre el modelo orientado a objetos y el modelo relacional y los problemas de concurrencia presentados en el desarrollo de la plataforma.
- Se diseñó e implementó una capa de acceso concurrente a Bases de Datos distribuidas permitiendo la interacción de las aplicaciones MPPP de la plataforma con la base de datos para la gestión de los módulos de cómputo paralelo.
- Se puso en práctica una alternativa de distribución de datos mediante réplica en Bases de Datos PostgreSQL para mejorar el rendimiento y la tolerancia a fallos del clúster de Bases de Datos.
- Se aplicaron técnicas de alta disponibilidad para garantizar la continuidad del servicio de acceso a datos y mejorar el nivel de tolerancia a fallos del clúster.

Recomendaciones

A partir de la investigación realizada se recomienda aplicar las experiencias alcanzadas en entornos de desarrollo similares. Continuar el desarrollo del Motor de Persistencia simultáneamente al resto de las aplicaciones de la plataforma para cubrir los nuevos requisitos de fases posteriores de desarrollo, como el soporte a varias Bases de Datos, tipos de datos distribuidos, creación de componentes distribuidos para la manipulación de datos y paralelización de consultas. Aplicar las técnicas propuestas para la optimización del diseño del modelo de datos y elaboración de consultas en la construcción de los módulos de cómputo para mejorar su desempeño y el acoplamiento a la plataforma. Extender el proceso de mejoras de pruebas para establecer propuestas de configuración del clúster de bases de datos para diferentes entornos. Aplicar las alternativas de configuración para clúster de alta disponibilidad para proyectos con necesidades similares.

Bibliografía

[1] K. Wolfgang.(2004)Persistence Options for Object-Oriented Programs. Disponible en: http://www.objectarchitects.de/ObjectArchitects/events/OOP2004/PersistenceOptionsOOP2004e.pdf. [Consultado en Marzo 2009].

[2] Sitio Oficial de Hibernate. Disponible en : http://www.hibernate.org [Consultado en Abril 2009]

[3] A. Miguel, M. Piattini. Fundamentos y Modelos de Bases de Datos. RA-MA. 1999. 456pp

[4] Colectivo de Autores. Diccionario de la Real Academia Española. Disponible en: http://www.rae.es/rae.html. [Consultado en Abril 2009]

[5] P. Boronat, M. Francisco. Concurrencia y Sistemas Distribuidos. 2003. 348 pp

[6] C. Evrendilek, et.al. A Preprocessor Approach to Persistent C++, Software Research and Development Center Scientic and Technical Research Council of Türkiye. 2005.

[7] Booch, G. "Análisis y Diseño Orientado a Objetos con Aplicaciones". 2a Ed. Addison- Wesley/Díaz de Santos. 1996, Vol. 20

[8] P. Pizarro. Arquitectura de Software: ORM, Object-Relational Mapping - I Parte. 2005. Disponible en: http://arquitectura-de-software.blogspot.com/2006/05/orm-object-relationalmapping-i-part.html. [Consultado en Febrero 2009]

[9] I. Jacobson, et al. Proceso Unicado de Desarrollo de Software. Madrid, Addison-Wesley. 2000. 458 pp.

[10] M. Townsend. Exploring the Singleton Design Pattern. 2002. Disponible en: http://msdn.microsoft.com/en-us/library/ms954629.aspx. [Consultado en Abril 2009]

[11] J. Sklenar. Introduction to OOP in Simula. 2007. Disponible en: http://staff.um.edu.mt/jskl1/talk.html. [Consultado en Abril 2009]

[12] J. Soulie. History of C++. 2008 Disponible en: http://www.cplusplus.com/info/history. [Consultado en Abril 2009]

[13] J. Byous. Java technology. The early years. Sun Developer Network, 2003. Disponible en: http://java.sun.com/features/1998/05/birthday.html. [Consultado en Abril 2009]

[14] Universidad Sevilla. Modelo relacional de Codd. Estructuras y restricciones. 2004 Disponible en: http://www.lsi.us.es/docencia/get.php?id=3183 [Consultado en Abril 2009]

[15] A. Alberca, J. Galvez. Modelos Avanzados de Bases de Datos. Universidad de Castilla, La Mancha, 2008. 102pp.

[16] Sitio Oficial de db4o Object Database. Disponible en: http://www.db4o.com/s/objectdb.aspx. [Consultado en Abril 2009]

[17] B. Stroustrup. 2009. Bjarne Stroustrup's homepage. Disponible en: http://www.research.att.com/~bs/. [Consultado en Abril 2009]

[18] D. Dagum. Objetos, Tablas Relacionales y Desajuste por Impedancia (Impedance Mismatch) 2007. Disponible en: http://diegumzone.spaces.live.com/blog/cns!1AD5096D63670065!307.entry [Consultado en Abril 2009]

[19] E.F. Codd. "A relational model for large shared data banks". Comunications of the ACM. 1970. Vol. 13, Páginas: 377 - 387 ISSN:0001-0782.

[20] D. Dagum. Mapeo de Objetos y Tablas Relacionales (O/R-M). Lo Que a Mí me Sirvió. 2008. Disponible en: http://diegumzone.spaces.live.com/Blog/cns!1AD5096D63670065!705.entry. [Consultado en Abril 2009]

[21] T. Newards. The Vietnam of Computer Science. 2006. Disponible en: http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx [Consultado en Marzo 2009]

[22] Sitio Oficial Debea: Database Acces Library. Disponible en: http://debea.net/trac [Consultado en Abril 2009]

[23] Sitio Oficial Simple Oracle Call Interface Disponible en: http://soci.sourceforge.net/ [Consultado en Abril 2009]

[24] Sitio Oficial Database Template Library Disponible en: http://dtemplatelib.sourceforge.net/ [Consultado en Abril 2009]

[25] Sitio Oficial LiteSql Disponible en: http://apps.sourceforge.net/trac/litesql/ [Consultado en Abril 2009]

[26] M. Reingart. PyReplica. 2008. Disponible en: http://www.arpug.com.ar/pgday2008/slides/pyreplica.pdf [Consultado en Enero 2009].

[27] E. Mustain. Introducing Slony. 2004. Disponible en: http://www.onlamp.com/pub/a/onlamp/2004/11/18/slony.html. [Consultado en Diciembre 2008]

[28] J. P. Paredes. Alta disponibilidad para Linux. 2001. Disponible en: http://www.ibiblio.org/pub/linux/docs/LuCaS/Presentaciones/200103hispalinux/paredes/pdf/LinuxHA.pdf. [Consultado en Marzo 2009]

[29] PGAdmin.org. The Artistic License. 2008. Disponible en: http://www.pgadmin.org/docs/1.8/licence.html. [Consultado en Marzo 2009]

[30] Sitio Oficial Proyecto UltraMonkey. Disponible en: http://www.ultramonkey.org/ [Consultado en Abril 2009]

[31] J. Rief. Ldirectord.200. Disponible en: http://www.vergenet.net/linux/ldirectord/ [Consultado en Abril 2009]

[32] Home of the Heartbeat project. Disponible en: http://moin.linux-ha.org/lha [Consultado en Abril 2009]

[33] E. Malinowski. Fragmentacion vertical de clases en las Bases de Datos distribuidas orientadas a objetos. 2000. Disponible en: http://www.dlsi.ua.es/asignaturas/bdd/articulos%20recomendados/Malinowski1999.pdf. [Consultado en Febrero 2009]

[34] P. Félix. Diseño e implementación de Bases de Datos. 2005 Disponible en: http://aluziner.googlepages.com/BD_Distribuidas.doc. [Consultado en Febrero 2009]

[35] C. J. Date. Introducción a los Sistemas de Bases de Datos. Pearson Prentice Hall, 2006

[36] Sitio oficial de Debian. 2009. Disponible en `http://www.debian.org/` [Consultado en Abril 2009]

[37] Eclipse C and C++ Development Tool. 2009. Disponible en: `http://www.easyeclipse.org/site/plugins/eclipse-cdt.html`. [Consultado en Abril 2009]

[38] libpq-C Library. 2009. Disponible en: `http://www.PostgreSQL.org/docs/8.3/interactive/libpq.html`. [Consultado en Abril 2009]

[39] Sitio Oficial de Visual-Paradigm UML. 2009. Disponible en: `http://www.visual-paradigm.com/`. [Consultado en Abril 2009]

Anexos

Anexo 1 Estándares de Diseño de Base de Datos.

Para la confección del modelo de datos de la plataforma Génesis se siguieron los siguientes estándares de diseño de Bases de Datos:

1. Nombres de los objetos.

   a) Los nombres de todos los objetos creados en la base de datos deben sugerir el significado de lo que representan en la vida real. Por ejemplo la tabla *tusuario* contiene la información de todos los *usuarios* del sistema. Se deben evitar términos ambiguos o que sugieran distintas interpretaciones.

   b) Todos los nombres se escriben con letra minúscula y para nombres compuestos se separan las palabras con el guión bajo "_" , por ejemplo *tnodo_servidor*, excepto para el caso de los identificadores que comienzan con "id" como el campo *idusuario* de la tabla *tusuario*. No deben abreviarse los nombres de los objetos.

   c) Únicamente se utilizarán caracteres alfabéticos, salvo que por la naturaleza del nombre se necesiten dígitos numéricos. Se prohíbe el uso de caracteres de puntuación o símbolos.

   d) Las letras acentuadas se reemplazarán con las equivalentes no acentuadas, y en lugar de la letra eñe (ñ) se utilizará (nn). Ejemplo: *descripcion* para la Descripción de un módulo de cálculo *y anno* para el Año de inscripción.

   e) Deben agregarse comentarios a las Bases de Datos y los campos, sobre todo a los booleanos.

2. Diseño de las tablas.

a) Los nombres de las tablas comienzan con una "t" y a continuación el nombre de la tabla que sugiere la interpretación del contenido que almacena, deben cumplir con las especificaciones anteriores.

b) Los primeros campos de cada tabla deben ser los que mayormente se utilizan, primero se definen los de longitud fija y después los de longitud variable para optimizar el proceso de consulta.

3. Campos de las tablas.

   a) Ajustar al máximo el tamaño de los campos para no desperdiciar espacio en la base de datos y seleccionar los tipos de datos apropiados.

   b) Identificar las restricciones de campos no vacíos (NOT NULL), de llaves primarias y claves compuestas.

   c) Los campos que representan la clave primaria deben ubicarse al inicio de las tablas para minimizar el tiempo en las operaciones de búsqueda.

   d) El nombre del campo clave debe estar compuesto por "id" + nombre de la tabla. Dependiendo de la naturaleza de la entidad, el nombre de la tabla a usar es el de la misma tabla, o el de la relacionada. Ejemplos: tabla *tusuario => idusuario.*

   e) Cuando en una tabla existen campos que tienen el mismo nombre, debe asociarse un alias que represente la información que almacenan. Ejemplo: En la tabla *tmensaje idusuario=> idusuario* que envía el mensaje e *idusuariodestino => idusuario* que recibe el mensaje.

4. Normalizar la base de datos hasta la tercera forma normal, para evitar redundancias en el almacenamiento. solo ceder en la normalización cuando esta puede ocasionar pérdida en la ejecución de una consulta.

Anexo 2 Instalación de PgCluster.

A continuación se describen los pasos para la configuración de la herramienta PgCluster para el clúster de Bases de Datos de la plataforma Génesis. Esta configuración es para entornos Linux.

Pasos para la instalación:

Requisitos previos:

Instalar los siguientes paquetes: zlib, openssl, openssh, rsync, PgCluster,

Configuración

En la configuración para la plataforma se utilizan 4 servidores llamados eipad1.uci.cu, eipad2.uci.cu, eipad3.uci.cu y eipad4.uci.cu respectivamente. La distribución del clúster es la siguiente:

Servidor de Balance de Carga: eipad1.uci.cu.
Servidor de Base de Datos 1: eipad2.uci.cu.
Servidor de Base de Datos 2: eipad3.uci.cu.
Servidor de Réplica: eipad4.uci.cu.

Creación del usuario postgres:

```
# adduser postgres
$ su -l postgres
$ set PGDATA=/usr/local/
$ export PGDATA
$ set LD_LIBRARY_PATH=/usr/local/pgsql/lib
$ export LD_LIBRARY_PATH
$ chown -R postgres /usr/local/pgsql
```

Inicialización de la base de datos:

La instalación del PgCluster se realiza en /usr/local/pgsql. Para crear una base de datos se debe inicializar la misma:

```
# su -l postgres
```

```
$ cd /usr/local/pgsql
$ bin/initdb -D /usr/local/pgsql/data -E EUC_JP –no-locale
```

Configuración de acceso al Clúster de Bases de Datos:

Las políticas de acceso se configuran en el fichero pg_hba.conf localizado en /usr/local/pgsql/data/ pg_hba.conf en cada uno de los servidores.

```
host all all 10.36.13.201 trust          #este es eipad1.uci.cu
host all all 10.36.13.202 trust          #este es eipad2.uci.cu
host all all 10.36.13.203 trust          #este es eipad3.uci.cu
host all all 10.36.13.204 trust          #este es eipad4.uci.cu
```

y en /usr/local/pgsql/data/PostgreSQL.conf

```
listen_addresses = '*'
port = 5432
max_connections = 100
```

Para permitir la resolución de los nombres de los servidores, en el archivo /etc/hosts de cada servidor:

```
10.36.13.201 eipad1.uci.cu eipad1
10.36.13.202 eipad2.uci.cu eipad2
10.36.13.203 eipad3.uci.cu eipad3
10.36.13.204 eipad2.uci.cu eipad4
```

Configuración del archivo pglb.conf localizado en /usr/local/pgsql/etc/pglb.conf para eipad1.uci.cu:

```
<cluster_server_info>
    <host_name>eipad2.uci.cu</host_name>
    <port>5432</port>
    <max_connect>32</max_connect>
</cluster_server_info>

<cluster_server_info>
    <host_name>eipad3.uci.cu</host_name>
```

```
    <port>5432</port>
    <max_connect>32</max_connect>
</cluster_server_info>

<host_name>eipad1.uci.cu</host_name>
<backend_socket_dir>/tmp</backend_socket_dir>
<receive_port>5433</receive_port>
<recovery_port>6001</recovery_port>
<max_cluster_num>128</max_cluster_num>
<use_connection_pooling>no</use_connection_pooling>
```

Configuración del archivo cluster.conf localizado en /usr/local/pgsql/ para eipad2.uci.cu:

```
<Replicate_Server_Info>
    <Host_Name>eipad4.uci.cu</Host_Name>
    <Port>8001</Port>
    <Recovery_Port>8101</Recovery_Port>
</Replicate_Server_Info>

<Recovery_Port>7001</Recovery_Port>
<Rsync_Path>/usr/bin/rsync</Rsync_Path>
<Rsync_Option>ssh -2</Rsync_Option>
<Rsync_Compress>yes</Rsync_Compress>
<Pg_Dump_Path>/usr/local/pgsql/bin/pg_dump</Pg_Dump_Path>
<When_Stand_Alone>read_only</When_Stand_Alone>
```

Configuración del archivo cluster.conf localizado en /usr/local/pgsql/ para eipad3.uci.cu:

```
<Replicate_Server_Info>
    <Host_Name>eipad4.uci.cu</Host_Name>
    <Port>8001</Port>
    <Recovery_Port>8101</Recovery_Port>
</Replicate_Server_Info>
```

<Recovery_Port>7001</Recovery_Port>
<Rsync_Path>/usr/bin/rsync</Rsync_Path>
<Rsync_Option>ssh -2</Rsync_Option>
<Rsync_Compress>yes</Rsync_Compress>
<Pg_Dump_Path>/usr/local/pgsql/bin/pg_dump</Pg_Dump_Path>
<When_Stand_Alone>read_only</When_Stand_Alone>

Configuración del archivo pgreplicate.conf creado manualmente en /usr/local/pgsql/etc para eipad4.uci.cu:

<Cluster_Server_Info>
    <Host_Name>eipad2.uci.cu</Host_Name>
    <Port>5432</Port>
    <Recovery_Port>7001</Recovery_Port>
</Cluster_Server_Info>

<Cluster_Server_Info>
    <Host_Name>eipad3.uci.cu</Host_Name>
    <Port>5432</Port>
    <Recovery_Port>7001</Recovery_Port>
</Cluster_Server_Info>

<Host_Name>eipad1.uci.cu</Host_Name>
<Replication_Port>8001</Replication_Port>
<Recovery_Port>8101</Recovery_Port>
<Rlog_Port>8301</Rlog_Port>
<Response_Mode>normal</Response_Mode>
<Use_Replication_Log>no</Use_Replication_Log>

Configuración de la autenticación automática con el fin de que los servidores se comuniquen entre sí:

Debe crearse el archivo “clavesautorizadas” en el home del usuario postgres en cada servidor:

```
# su -l postgres
$ ssh-keygen -t rsa
$ cd .ssh
$ cp identity.pub clavesautorizadas
```

Luego se concatena la clave generada en cada uno al archivo clavesautorizadas de manera que el mismo contenga las claves de ambos. En cada servidor se configura el archivo /etc/ssh/sshd_config con la siguiente información:

```
RSAAuthentication yes
PubkeyAuthentication yes
AuthorizedKeysFile .ssh/clavesautorizadas
```

Para arrancar los servicios:

Servidor de Réplica:

```
# su -l postgres
$ /usr/local/pgsql/bin/pgreplicate -D /usr/local/pgsql/etc
```

Servidor de Bases de Datos:

```
$ /usr/local/pgsql/bin/pg_ctl -D /usr/local/pgsql/data -o "-i" start
```

Servidor de Balance de Carga:

```
$ /usr/local/pgsql/bin/pglb -D /usr/local/pgsql/etc
```

Sugerencias:

1. Verificar que estén activos los demonios del ssh y rsync antes de correr los servicios replicadores y de Bases de Datos.

2. Eliminar los archivos temporales creados por los servicios de replicación y de Bases de Datos cuando estos sean reiniciados y antes de correrlos nuevamente.

Anexo 3 Instalación de HeartBeat.

A continuación se muestra la configuración de servicio de alta disponibilidad definido para el Servidor de Balance de Carga de la plataforma Génesis, mediante la utilización de la herramienta Heartbeat.

1. Se emplean dos servidores para garantizar la alta disponibilidad uno fucionará como servidor principal y el otro será su servidor espejo o alternativo.

   *Sevidor Principal: eipad1.uci.cu ip 10.36.13.201

   *Servidor Espejo: eipad5.uci.cu ip 10.36.13.205

   IP Virtual: 10.36.13.220 # En este ip se ofrece el servicio de alta disponibilidad.

2. Instalar HeartBeat: Para instalar este paquete desde un repositorio de Debian debe ejecutar la línea de comando: apt-get install heartbeat.

3. Editar el fichero hosts que se localiza en /etc/hosts.

   Contenido del fichero:

   127.0.0.1 localhost
   10.36.13.201 eipad1.uci.cu eipad1
   10.36.13.205 eipad5.uci.cu eipad2

4. En el Servidor Activo o principal se debe añadir en el fichero hosts la configuración del ip virtual.

   iface eth0:0 inet static
   address 10.36.13.220
   network 10.36.0.0
   netmask 255.255.255.0

5. Se deben modificar en ambos servidores tres ficheros que se encuentran localizados en /etc/ha.d.

   - /etc/ha.d/authkeys
   auth 1
   1 crc

   En authkeys se especifican las claves,en este caso se utiliza el método CRC (que tiene menos carga de CPU) asumiendo que se tiene un buen firewall configurado, podría usarse md5, o sha1 para mayor seguridad.

- /etc/ha.d/ha.cf

debugfile /var/log/ha-debug

logfile /var/log/ha-log logfacility local0

keepalive 2

deadtime 10

warntime 10

initdead 100

udpport 694

bcast eth0

ucast eth0 10.36.13.201

auto_failback on

node eipad2

node eipad4

En el fichero ha.cf se especifica las opciones de configuración de heartbeat, se indica donde se desea guardar los logs, en keepalive le indicamos que los latidos se enviarán cada 2 segundos, en deadtime especificamos que si un nodo no responde en 10 segundos está muerto, en warntime le decimos que si no responde en 10 segundos nos lanzará una alerta, en el initdead antes de considerar un nodo muerto esperaremos 100 segundos para evitar problemas con el arranque del servicio, usaremos el puerto UDP 694, ponemos el parámetro failback en on para indicar el comportamiento en caso de recuperación de un nodo caído, estando en on este recuperará todos los servicios al volver a estar activo si lo dejamos en off, los servicios residirán en el nodo espejo hasta que este caiga, finalmente indicamos los nodos forman el servicio y un dispositivo ethernet para broadcast y en ucast se debe especificar para cada nodo su ip.

/etc/ha.d/haresources

eipad1 IPaddr2::10.36.13.220/24/eth0 apache2 pgsql

En el fichero *haresources*, especificaremos los servicios que debe manejar heartbeat, este fichero es el mismo en los dos nodos. Solo se debe especificar el nombre del servidor.

6. Finalmente se reinicia heartbeat en ambos servidores con /etc/init.d/heartbeat restart.

Glosario de Términos

Clúster: Conglomerado de computadoras conectadas por una red de alta velocidad.

CRUD: Se refiere a los términos en inglés Create, Read, Update y Delete referidos a las operaciones de Crear, Recuperar, Actualizar y Eliminar respectivamente.

GPL: Licencia de Software Libre del proyecto GNU.

iODBC: Acrónimo de Independent Open DataBase Connectivity. Es una plataforma de código abierto que ofrece una implementación independientes de las especificaciones ODBC y X/Open.

MPPP: Motor de Procesamiento de Peticiones Paralelas acoplado a la plataforma Génesis para la gestión del cálculo de los módulos de cómputo.

MPPPC: Motor de Procesamiento de Peticiones Paralelas en modo cliente que se instala en los nodos esclavos del clúster de procesamiento de la plataforma.

MPPPS: Motor de Procesamiento de Peticiones Paralelas en modo servidor que se instala en el nodo servidor del clúster de procesamiento de la plataforma.

Nodo: En el área de la computación paralela representa una computadora en un clúster. Los clúster están compuestos por un conjunto de nodos.

ODBC: Acrónimo de Open Database Connectivity. Estándar de acceso a Bases de Datos desarrollado por Microsoft Corporation. Su objetivo es hacer posible el acceso a cualquier dato desde cualquier aplicación, sin importar qué Sistema Gestor de Bases de Datos almacene los datos.

ODL: Acrónimo de Object Definition Language. Lenguaje de definición de objetos para Bases de Datos Orientada a Objetos.

ODMG: Acrónimo de Object Database Management Group. Se usa tanto para definir el grupo de personas y empresas encargadas de desarrollar el modelo de objetos para persistencia, así como para la definición de dicho estándar. Especifica los elementos que se definirán, y en qué manera se hará, para la consecución de persistencia en las Bases de Datos Orientadas a Objetos que soporten el estándar.

OQL: Acrónimo de Object Query Language. Lenguaje de consultas entre objetos para Bases de Datos Orientadas a Objetos. Es la analogía del lenguaje SQL para el modelo relacional.

Página: Mensaje que contiene un dato que envía un nodo del clúster de procesamiento a otro. Los mensajes que llegan a un nodo cuando este no lo ha solicitado aún se paginan, es decir, son guardados en la base de datos por el Buffer de Mensajes que lo extrae cuando es solicitado.

Plataforma: Conjunto de herramientas, librerías, documentación para el desarrollo de aplicaciones de software.

Semáforo: Variable especial protegida que constituye el método clásico para restringir o permitir el acceso a recursos compartidos en un entorno de multiprocesamiento donde se ejecutarán varios procesos concurrentemente.

Thread-safety: Término del inglés muy utilizado en la programación multihilo. Una pieza de código es "thread-safety" si funciona correctamente durante la ocurrencia simultánea de múltiples hilos de ejecución.