Universidad de las Ciencias Informáticas

Facultad 9

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

“Propuesta de una solución de replicación para el gestor de bases de datos PostgreSql en la Oficina Nacional de Recursos Minerales.”

Autor:

Rider Masó Fonseca.

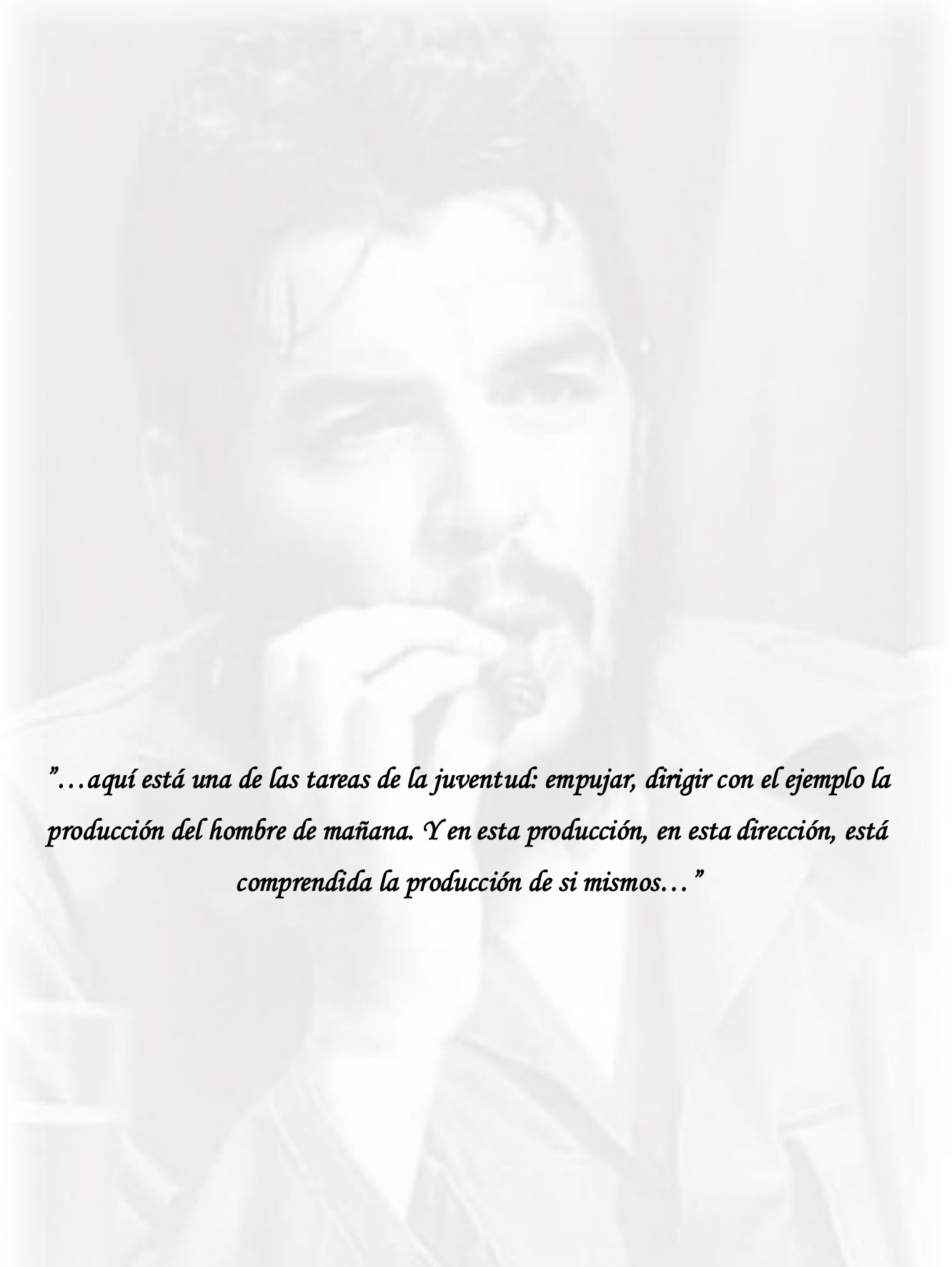
Tutores:

Lic. Ridosbey Milián Iglesias.

Co-Tutor:

Ing. Kenny Rodríguez García.

Ciudad de la Habana, 2010



"…aquí está una de las tareas de la juventud: empujar, dirigir con el ejemplo la producción del hombre de mañana. Y en esta producción, en esta dirección, está comprendida la producción de si mismos…"

Declaración de Autoría

Declaramos que somos los únicos autores del trabajo titulado:
Propuesta de una solución de replicación para el gestor de base de datos PostgreSql en la Oficina Nacional de Recursos Minerales, y autorizamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los días del mes de julio del año 2010.

Rider Masó Fonseca
<Nombre del autor>

Lic. Ridosbey Milián Iglesias
Nombre del primer Tutor

Ing. Kenny Rodríguez García
Nombre del segundo Tutor

Agradecimientos

A esta Revolución por permitirme estudiar en esta maravillosa universidad y en especial a nuestro eterno Comandante Fidel protagonista de que este sueño se hiciera realidad.

A mis tutores Ridosbey y Kenny por sus consejos y ayuda. Gracias por ser tan exigentes, pacientes y por depositar su confianza en mí.

A mi madre en especial que lo es todo para mí y a toda mi familia en general, gracias por creer en mí.

A Lidia y a Rafael gracias por su preocupación durante los interminables días de tensión.

A Leidy por su apoyo incondicional que con tanto interés siguió de cerca el trabajo, gracias por tus consejos y dedicación.

A Gerdys y Eddy que en su momento me dieron sus consejos para mejorar la tesis.

A todos mis amigos del ajedrez a los que tanto estimo y por tantos momentos que pasamos juntos.

Resumen

En el presente trabajo de diploma se realiza un análisis del estado del arte sobre cómo se realiza el proceso de réplica en una base de datos distribuida así como el estudio y caracterización de un conjunto de herramientas libres de replicación.
Con esta investigación se logró como resultado una solución de replicación robusta para PostgreSql que garantiza la integridad y la disponibilidad de los datos en la Oficina Nacional de Recursos Minerales, se describen el entorno de replicación y las características que debe tener la propuesta para satisfacer las necesidades de dicha institución.
El proceso de despliegue se lleva a cabo en la Oficina Nacional de Recursos Minerales demostrando cómo se pueden replicar datos en un ambiente real de producción. Para ello se evalúan las condiciones de esta institución de acuerdo a la distribución de su entorno de desarrollo y los recursos que dispone proponiendo un modelo de implantación de final.

Con esta solución se logra mayor calidad y visibilidad en el proceso de replicación de las bases de datos pues con este resultado se garantiza una alta disponibilidad y un elevado rendimiento en los servidores de bases de datos de la Oficina Nacional de Recursos Minerales y el Ministerio de la Industria Básica al contar con medidas cuantitativas que permiten mayor rendimiento, disponibilidad y confiabilidad en los datos.

Palabras claves: Réplica, Base de datos.

Índice de Contenido

Índice de Figuras

Introducción

La Oficina Nacional de Recursos Minerales (ONRM), constituye la autoridad minera en la República de Cuba y tiene como misión principal garantizar el aprovechamiento racional de los recursos minerales del país. Su creación queda estipulada en la Ley No.76 (“Ley de Minas”) el 23 de enero de 1995, constituida como una institución con personalidad jurídica adscrita al Ministerio de la Industria Básica (MINBAS). Esta importante entidad geológica se encarga de controlar y fiscalizar el proceso concesionario, garantizar la protección de los recursos minerales y de hidrocarburos, además, de ejercer con rigor técnico el control estatal de la explotación racional de los recursos minerales y la preservación del medio ambiente durante el desarrollo de las actividades de la geología, minería y del petróleo.

La ONRM es depositaria de grandes volúmenes de información que hacen referencia al conocimiento geológico del país acumulado durante siglos. Se cuenta hoy con un archivo técnico donde se recibe, organiza y conserva este patrimonio documental y está en función de dar servicio a toda la comunidad de las Geociencias. Un aumento significativo en la agilidad y efectividad del trabajo en la oficina lo representó la introducción de los primeros medios de computación en el campo de las Ciencias Geológicas en 1997. En este año se logró la elaboración de la base de datos digital del Balance y de las Concesiones Mineras, más adelante, la digitalización de datos de los principales recursos minerales como níquel y petróleo, levantamientos aerogeofísicos nacionales, mapas geológicos a diferentes escalas, más de 100 bases de datos de las principales zonas de interés económico, estructuradas y documentadas.

En la actualidad en la ONRM existe un servidor de Base de Datos centralizado que es donde se encuentra toda la documentación referente a esta entidad; por otra parte el MINBAS cuenta con otros dos servidores de Base de Datos (Servidor de Mapas), y otro servidor con las bases de datos de las aplicaciones que están publicadas en Internet, localizados en una red externa a la ONRM. El Ministerio como entidad rectora necesita mantener actualizados ciertos datos provenientes de las bases de datos de la Oficina Nacional, este proceso se lleva a cabo manualmente, haciendo engorroso un trabajo que actualmente puede ser automatizado, al duplicar los esfuerzos de las personas encargadas de esta actividad, además de los viajes reiterados de una sede a otra.

La replicación es un mecanismo utilizado para propagar y diseminar datos en un ambiente distribuido, con el objetivo de obtener mejor rendimiento y confiabilidad, mediante la reducción de dependencia de un sistema de base de datos centralizado [1].

Las instituciones utilizan la replicación en sus aplicaciones por varias razones, las cuales pueden ser categorizadas de la siguiente forma:

- Distribución de datos a otras ubicaciones.
- Consolidación de datos desde otras ubicaciones.
- Intercambio bidireccional de datos con otras ubicaciones.

La replicación de datos tradicional es asequible, pero tan sólo ofrece acceso a una copia de datos en tiempo real en el momento de la caída y no trata con los factores de disponibilidad necesarios para mantener a los usuarios conectados a sus aplicaciones. Por otro lado, las tecnologías de clustering (conjunto de ordenadores que se comportan como si fuera una sola computadora) tradicionales ofrecen alta disponibilidad, pero son demasiado costosas y complejas de instalar y manejar, particularmente a través de una red WAN[1] [1].

Los mecanismos de réplica presentan problemas cuando se trata de este tipo de redes, incluso en Gestores de Bases de Datos tan utilizados como SQLServer reconocido mundialmente por su escalabilidad, estabilidad y seguridad, es decir, cuando hay aplicaciones grandes a nivel de redes WAN, los mecanismos de réplicas presentan problemas y hay que recurrir a soluciones a nivel de redes para que exista comunicación entre los mecanismos de réplica establecidos. PostgreSql no está exento de este problema.

Tomando en cuenta la situación actual se hace necesario que las tecnologías a aplicar en los servidores de bases de datos de la nueva solución satisfagan las necesidades funcionales de la empresa.

Por lo antes expuesto se hace necesario plantear el siguiente **problema a resolver:**

¿Cómo realizar una solución de replicación para PostgreSql que garantice la integridad y la disponibilidad de los datos en la Oficina Nacional de Recursos Minerales a través de redes WAN?

Para dar solución al problema planteado se define como **objeto de estudio** el proceso de réplica para bases de datos distribuidas y se tiene como **objetivo general,** proponer una solución de replicación de base de datos para el gestor PostgreSql que garantice la integridad y la disponibilidad de los datos en la Oficina Nacional de Recursos Minerales.

[1] Una Red de Area Amplia, es un tipo de red de computadoras capaz de cubrir distancias desde unos 100 km hasta unos 1000km, dando el servicio a un país o un continente.

Para lograr el objetivo general se plantean los siguientes **objetivos específicos:**

- Realizar un análisis de las tendencias actuales en la utilización de la réplica de bases de datos.
- Identificar y caracterizar los entornos de réplica sobre PostgreSql.
- Realizar una propuesta de solución de réplica sobre PostgreSql en la Oficina Nacional de Recursos Minerales.

Se precisa como **campo de acción** el proceso de réplica de bases de datos distribuidas en PostgreSql.

Atendiendo a los objetivos planteados se desarrollarán las siguientes **tareas de la investigación.**

- Analizar las tendencias actuales en la utilización de la réplica de bases de datos en el mundo y en Cuba.
- Analizar la evolución de las herramientas de réplica de bases de datos para PostgreSql en el mundo y en Cuba.
- Identificar y caracterizar los entornos de réplica sobre PostgreSql.
- Proponer una solución de replicación robusta que garantice la integridad de los datos y la independencia tecnológica en la ONRM.
- Elaborar una guía para el montaje del entorno de replicación haciendo uso de la solución que se pretende aplicar.
- Realizar pruebas a la solución de réplica seleccionada en entornos de producción.
- Presentar los resultados.

Los métodos científicos usados en la presente investigación se describen en la siguiente tabla:

Métodos Científicos	**Descripción**	**Utilización**
Métodos Teóricos		
Histórico Lógico	Analizan la trayectoria completa del fenómeno, su condicionamiento a los diferentes períodos de la	Permitió conocer la evolución de las herramientas de replicación de bases de datos en el mundo y en

	historia, revela las etapas principales de su desenvolvimiento y las conexiones históricas fundamentales [2].	Cuba.
Analítico Sintético	Permite la división mental del fenómeno en sus múltiples relaciones y componentes para facilitar su estudio y establece mentalmente la unión entre las partes previamente analizadas, posibilitando descubrir sus características generales y las relaciones esenciales entre ellas [2].	• Permitió determinar la importancia de una solución de réplica robusta que garantice la integridad y la disponibilidad de los datos en la Oficina Nacional de Recursos Minerales que se extiendan sobre redes WAN. .
Hipotético-Deductivo	Consiste en deducir y explicar leyes e hipótesis de menor nivel de generalidad y abstracción a partir de propuestas de mayor nivel de generalidad, abstracción y lógica [2].	• Permitió deducir cómo una solución de réplica para PostgreSql robusta, garantiza la integridad y la disponibilidad de los datos en la Oficina Nacional de Recursos Minerales que se extiendan sobre redes WAN.

Resultados esperados

Los resultados que se esperan de la presente investigación son:

• Contar con una solución de replicación robusta para PostgreSql que garantice la integridad y la disponibilidad de los datos en la Oficina Nacional de Recursos Minerales que se extiendan sobre redes WAN.

Estructura de la investigación

El **Capítulo 1 Fundamentación Teórica** presenta un análisis del estado del arte sobre los mecanismos de réplica para PostgreSql utilizados en el mundo y las tendencias sobre la réplica de datos en Cuba. Para lograr los objetivos de esta fase se expone un análisis sobre qué es la replicación de datos, atendiendo a sus elementos principales y formas de uso. Se realiza además una caracterización de las principales herramientas de réplica para PostgreSql.

En el **Capítulo 2 Propuesta de Solución** se realiza un estudio detallado sobre la solución de réplica propuesta. Para ello se plantea como podrá utilizarse y los requerimientos necesarios para ello.

En el **Capítulo 3 Aplicación de SymmetricDS como solución de réplica en la Oficina Nacional de Recursos Minerales** se presentan los resultados obtenidos de la investigación, a partir del cumplimiento de los objetivos propuestos.

Capítulo 1: Fundamentación Teórica.

En este capítulo se realiza un estudio acerca de las tendencias actuales sobre la replicación de datos en el mundo y los gestores de bases de datos que implementan herramientas de réplica. Se hace una exposición sobre los términos software libre y propietario, haciendo énfasis de la necesidad del uso de software libre en Cuba. En esta dirección, la investigación se centra en el uso de PostgreSql en el país y las principales herramientas de replicación que lo soportan, realizando para ello una explicación detallada de cada una de ellas.

1.1 Estado del arte.

El uso de las bases de datos (BD) en el mundo se ha extendido para facilitar mucho más la vida de los usuarios, lo cierto es que cuando se menciona “base de datos” viene de repente a la mente el concepto de SQL (lenguaje declarativo de acceso a BD), pues siempre ha sido la primicia fundamental para el manejo de datos, desde el punto de vista informático.

Hoy día existen varios gestores de BD que soportan la mayoría de la información que se maneja en el mundo, como es el caso de SQLServer, SQLite, Oracle, MySQL y PostgreSql, a partir de esta información se construyen gran parte de las aplicaciones informáticas existentes, tanto a nivel de usuario como de organizaciones de cualquier tamaño [3].

Con el transcurso del tiempo la información ha ido incrementándose vertiginosamente en el mundo, cada vez son más las empresas que mantienen sistemas muy complejos donde la información es accedida y modificada por usuarios potenciales y estos a su vez se incrementan día a día, por lo que mantener una sola BD centralizada acarrea problemas de rendimiento, aumentando los tiempos de respuesta, ocurriendo en ocasiones fallas en los servicios por lo que la información deja de estar disponible.

Es por esta razón que los desarrolladores se dieron a la tarea de implementar algoritmos que permitieran la disponibilidad de los datos en todo momento. De esta manera surge la replicación de datos y con ella la introducción de herramientas de réplica para cada gestor de BD.

En SQLServer por ejemplo, se utilizan tres tipos de replicación fundamentalmente: *Replicación Instantánea (Snapshot Replication)*, *Replicación por transacción (Transactional Replication),* y *Replicación mezclada (Merge Replication)*. Tanto el primero como el segundo se basan en un modelo de replicación en una sola dirección, desde un único publicador hacia los suscriptores, mientras que el

último permite que los diferentes servidores actúen con alto nivel de independencia y desconectados entre sí [4].

Oracle, por su parte soporta varios tipos de replicación como por ejemplo: *Vistas materializadas (materialized views), sólo lectura y actualizables*, que se encargan de duplicar de forma instantánea los datos de un sitio maestro en otros sitios; y la *Replicación avanzada*, la cual replica la información en varios sitios maestros.

Como se explicó con anterioridad la replicación se utiliza para distribuir información hacia otras ubicaciones así como el intercambio bidireccional con otras instituciones, esto se hace a través de la red usando una solución de replicación con el objetivo de propagar los datos, para lograr una mejor comprensión del tema hay que explicar qué tipos de redes existen y cómo se utilizan en la replicación de datos. Red de área local o red LAN es la que se utiliza dentro de un mismo edificio, es decir, para cubrir distancias pequeñas en un entorno de 200 metros hasta 1 kilómetro, y la red de área amplia o red WAN es para cubrir distancias grandes que pueden cubrir desde 100 kilómetros hasta 1000 kilómetros.

En el caso de la replicación de datos en una red LAN es importante, pues toda la información está distribuida en los servidores centralmente, pero la principal desventaja radica en que si se quiere replicar datos hacia una institución que esté adjunta a la misma pero no se encuentra en el mismo entorno o localidad física van a existir problemas con este tipo de redes y entonces hay que recurrir a soluciones de redes de tipo WAN para lo cual haría falta una herramienta para replicar que posea protocolos http[1] o https[2] para enviar los datos ya sea a través de internet o de una entidad a otra.

En la actualidad el mundo informático se encuentra dividido en dos grandes grupos: los que venden licencias para el uso de determinados software y los llamados productores de software libre. Los gestores de bases de datos también se sitúan en estas dos corrientes.

Antes de entrar en detalles, es bueno explicar de manera sencilla en qué consiste cada una de estas dos tendencias:

El **software no libre,** que también es llamado *software propietario*, *software privativo*, *software privado* o *software con propietario*. Se refiere a cualquier programa informático en el que los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo (con o sin modificaciones), y que su código fuente no está disponible o el acceso a éste se encuentra restringido.

[1] Protocolo destinado a la transferencia de datos hipertexto.
[2] Protocolo destinado a la transferencia segura de datos hipertexto.

En el software no libre una persona física o jurídica (corporación o fundación) posee los derechos de autor sobre un programa negando o no otorgando, al mismo tiempo, los derechos de usarlo con cualquier propósito; de estudiar cómo funciona y adaptarlo a las propias necesidades (donde el acceso al código fuente es una condición previa); de distribuir copias; o de mejorarlo y hacer públicas las mejoras (para esto el acceso al código fuente es un requisito previo). De esta manera, un software sigue siendo no libre aún si el código fuente es hecho público, cuando se mantiene la reserva de derechos sobre su uso, modificación o distribución [5].

Por su parte el **software libre** es aquel que puede ser distribuido, modificado, copiado y usado. Por lo tanto, debe venir acompañado del código fuente para hacer efectivas las libertades que lo caracterizan. Pero es necesario tener en cuenta algunos aspectos que tienden a confundir. Por ejemplo, el software de dominio público significa que no está protegido por el copyright (derecho de autor), por lo tanto, podrían generarse versiones no libres del mismo. En cambio el software libre protegido con copyleft (contrapartida del copyright) impide a los redistribuidores incluir algún tipo de restricción a las libertades propias del software así concebido, es decir, garantiza que las modificaciones seguirán siendo software libre. También es conveniente no confundir el software libre con el software gratuito (este no cuesta nada, hecho que no lo convierte en software libre), porque no es una cuestión de precio, sino de libertad, como lo plantea Richard Stallman, quien define cuatro clases de libertades para los usuarios del software [5]:

- **La libertad 0:** Se refiere a la libertad para ejecutar el programa con cualquier propósito.
- **La libertad 1:** Que permite además, estudiar y modificar el programa.
- **La libertad 2:** Permite copiar el programa de manera que puedas ayudar a tu vecino.
- **La libertad 3:** Brindando la posibilidad de mejorar el programa, y hacer públicas tus mejoras, de forma que se beneficie toda la comunidad.

Aún así, el término software libre se interpreta erróneamente en este sentido, por lo que, en contraposición a él, nace el llamado *open source (código abierto)*, que pone énfasis en que el código fuente está disponible para su consulta, modificación y distribución.

Lo cierto es que cada vez más los programas Open Source están ganando terreno en el mundo y proporcionan de cierta forma mejores divisas tanto para los clientes, como para los creadores o desarrolladores.

En los últimos años, las soluciones libres han alcanzado un grado de madurez similar a los prestados por los sistemas propietarios en cuanto a eficiencia y seguridad. Uno de los ejemplos más fehacientes es el auge que ha alcanzado el gestor PostgreSql, el cual se encuentra en la puntera de los Sistemas Gestores de Bases de Datos por su robustez e inclusión de características avanzadas similares a las incluidas en los sistemas propietarios y que no se encuentran en muchas otras soluciones libres [3].

El uso de PostgreSql se ha generalizado globalmente, siendo el gestor más utilizado por las grandes empresas desde el año 2008. Cada una de ellas ha tenido que distribuir la información, utilizando mecanismos de réplicas sobre este gestor adecuados y adaptados a sus condiciones. Algunas de ellas son [6]:

1. **HSBC (banco):** Brinda servicios financieros a través de la red.

2. **General Electric (bienes):** Es una empresa americana multinacional de infraestructuras, servicios financieros y medios de comunicación altamente diversificada. Desde energía, agua, transporte y salud hasta servicios de financiación e información.

3. **Bank of America (banco):** Es la banca comercial más grande de los Estados Unidos de América en cuanto a depósitos, y la compañía más grande del mundo en su categoría.

4. **JPMorgan Chase (banco):** Empresa americana considerada una de las empresas de servicios financieros más antiguas del mundo, líder en inversiones bancarias, servicios financieros, gestión de activos financieros e inversiones privadas.

5. **ExxonMobil (petróleo):** Empresa americana cuyas actividades se extienden por más de 40 países de todo el mundo e incluyen, entre otras, la explotación, elaboración y comercialización de productos petroleros y gas natural, así como la fabricación de productos químicos, plásticos y fertilizantes.

6. **Royal Dutch Shell (petróleo):** Compañía Real Holandesa de hidrocarburos que tiene intereses en los sectores petrolíferos, del gas natural así como del refino de gasolinas.

7. **BP (petróleo):** BP es una compañía de energía, dedicada principalmente al petróleo y al gas, que tiene su sede en Londres, Reino Unido. Es una de las mayores compañías del mundo y la tercera empresa más importante dedicada al petróleo y gas después de ExxonMobil y Royal Dutch Shell.

8. **Toyota Motor (bienes):** Empresa que se dedica a la venta de automóviles convertida en una de las empresas japonesas más rentables y con más éxito y una de las corporaciones líderes en la industria del automóvil.

9. **AT&T (Comunicaciones):** Es una empresa de telecomunicaciones de primer nivel en los Estados Unidos y a nivel mundial, con subsidiarias que operan y brindan servicios bajo la marca de AT&T, esta última reconocida como líder mundial en la provisión de servicios de comunicaciones basadas en IP para empresas y como el proveedor líder en los Estados Unidos de servicios celulares, servicios de acceso a Internet de alta velocidad, servicios telefónicos de voz locales y de larga distancia, así como en publicación de directorios y servicios publicitarios.

En Cuba el uso de la Informática ha estado soportado por el sistema operativo Windows. Sin embargo, actualmente el sistema LINUX y sus distintas distribuciones, han ganado espacios como sistema libre y de código abierto lo cual le confiere numerosas ventajas. Por esta razón el país se prepara para realizar la completa migración hacia este sistema.

Las experiencias cubanas en el software libre han permitido conseguir logros, como por ejemplo el desarrollo de la versión 2.0 del sistema operativo Nova, impulsado por la presentación de este software en la Convención Internacional Informática 2009, reafirmando que el país apuesta por la independencia tecnológica en esta rama.

Tal y como las grandes empresas mundiales, en Cuba los proyectos de desarrollo de software están potenciando el uso de PostgreSql como gestor de bases de datos. La Universidad de las Ciencias Informáticas (UCI) es un ejemplo de estas empresas, donde se ha creado además, la Comunidad Cubana de PostgreSql, en la que un numeroso grupo de profesionales y estudiantes se han especializado en proveer soluciones integrales y consultorías relacionadas con tecnologías de bases de datos y análisis de información. Por otro lado se encuentra el Centro de Desarrollo de Software de Villa Clara, que es una pequeña institución perteneciente a la Universidad de las Ciencias Informáticas cuya misión es la de potenciar herramientas para fortalecer este gestor.

Tras la creación de complejos sistemas de software donde intervienen grandes cantidades de usuarios y donde la información necesita estar disponible todo el tiempo, se ha venido abogando por la utilización de soluciones de réplica que permitan garantizar la integridad de los datos que se manejan en estos sistemas. Por tal motivo, en la UCI se han realizado investigaciones donde se ponen de manifiesto, de acuerdo a las características de cada entorno, elementos que tributan a qué herramienta es la más eficiente en cada caso. Dentro de estas investigaciones se encuentra el *"Sistema de réplica para la Intranet Corporativa y el Sitio en Internet de PDVSA."*, en el cual emplean una solución de réplica sobre PostgreSql.

Es bueno destacar en este punto que este gestor posee varias herramientas de replicación, las cuales serán detalladas en otros epígrafes, sin embargo las más usadas, no presentan todas las

potencialidades necesarias y en muchas ocasiones frenan a los desarrolladores a usarla en soluciones empresariales grandes, como es el caso de los Proyectos Productivos Registros y Notarias e Identidad de la propia universidad, los cuales eligieron Oracle principalmente por su soporte y por poseer una réplica de datos muy confiable.

1.2 El Gestor PostgreSql.

En este epígrafe se detallan los aspectos esenciales relacionados con el gestor PostgreSql, pero para comenzar a explicar este tema es importante tener en cuenta algunos conceptos fundamentales para su comprensión.

1.2.1 ¿Qué es una base de datos?

Una base de datos (BD) es el lugar donde se agrupan un conjunto de datos organizados que pertenecen a un mismo contexto. En la actualidad, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos tienen formato electrónico, que ofrece un amplio rango de soluciones al problema de almacenar datos [7].

Por definición, una base de datos es un objeto estructurado. Puede ser un montón de papeles, pero actualmente es más probable encontrarlas en sistemas informáticos. Esa estructura se compone de objetos de datos y metadatos, los metadatos no son más que la estructura. Los datos en una BD almacenan la información descriptiva. Metadatos describe la estructura aplicada por la base de datos a los datos. Se aplica la estructura y la organización de los datos brutos [8].

Es importante destacar que para un mejor entendimiento sobre la investigación el primer concepto está más completo que el otro que se expone pues está más enfocado a la informática y al tema de la replicación en específico.

1.2.2 Sistemas Gestores de Bases de Datos.

Los sistemas gestores de bases de datos (SGBD) surgen para el manejo de los datos en una BD, permitiendo almacenarlos y posteriormente acceder a los datos de forma rápida y estructurada.

El objetivo fundamental de un SGBD consiste en suministrar al usuario las herramientas que le permitan manipular, en términos abstractos, los datos, o sea, de forma que no le sea necesario conocer el modo de almacenamiento de los datos en la computadora, ni el método de acceso empleado [9].

Gestor PostgreSql.

PostgreSql es un Sistema de gestión de BD relacional orientada a objetos, se distribuye como software libre y bajo licencia BSD.

Este gestor ha ganado una importante comunidad de seguidores a nivel mundial la cual lleva por nombre PostgreSql Global Development Group (PGDG). Es importante destacar en este punto que América Latina posee una fuerte influencia en las mejoras realizadas a PostgreSql, puesto que posee 8 desarrolladores de los 25 existentes a nivel mundial que guían el desarrollo del mismo [10].

PostgreSql es compatible con ANSI SQL[1] , y se provee de características que hace posible un diseño de software más complejo. PostgreSql es extremadamente modular, soporta un gran número de tipos de datos, y es compatible hoy día un gran número de interfaces de programación. PostgreSql está referenciado por los lenguajes de programación más importantes, incluyendo C, Perl, Python, Tel, Java, PHP, ODBC, JDBC, entre otros. PostgreSql da soporte para la herencia y la seguridad de la capa de dispositivo de transportación de datos (SSL, Secure Sockets Layer). Además cumple completamente con las características ACID (acrónimo de Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español) para realizar transacciones seguras, es multiplataforma, está disponible para 34 plataformas en su última versión estable [11].

Muchas son las características de este gestor de bases de datos que tienen un impacto decisivo para la determinación de su uso frente a otros sistemas como Oracle, MySQL, SQL Server, etc. Lo primero que se debe tener en cuenta es la extensibilidad de su código que permite a los grupos de desarrollo hacer mejoras dentro de él (término en inglés: open source), de ahí su fuerte comunidad de seguidores. Desde el punto de vista del desarrollo se pueden citar algunas características importantes como:

- **Alta concurrencia:** donde mediante el acceso concurrente multiversión permite que mientras un proceso escribe en una tabla, otros accedan a la misma sin necesidad de bloqueos [10].

- **Creación de funciones en varios lenguajes:** donde se pueden realizar condicionales, bucles, así como aplicar los paradigmas de la programación orientada a objetos.

- **Diseñado para ambientes de alto volumen de información:** a través del uso de la estrategia de almacenamiento de acceso concurrente multiversión de filas (siglas en inglés MVCC) se puede conseguir un tiempo de respuesta mejor en ambientes que posean grandes volúmenes de datos.

1 Lenguaje estructurado de consultas.

- **Replicación:** que permiten la duplicación de bases de datos maestras en múltiples sitios de réplica, etc.

1.3 Replicación de Bases de datos.

La replicación es el proceso de copiar y de mantener los objetos de la BD en los múltiples nodos de BD que incorporan un sistema de base de datos distribuida [9].

Una **base de datos distribuida (BDD)** es un conjunto de múltiples bases de datos lógicamente relacionadas las cuales se encuentran distribuidas entre diferentes sitios interconectados por una red de comunicaciones, los cuales tienen la capacidad de procesamiento autónomo lo cual indica que puede realizar operaciones locales o distribuidas [11].

La diferencia principal entre los sistemas de base de datos centralizados y distribuidos es que, en los primeros, los datos residen en una sola localidad, mientras que, en los últimos, se encuentran en varias localidades.

Un **Sistema de Bases de Datos Distribuida (SBDD)** es un sistema en el cual múltiples sitios de bases de datos están ligados por un sistema de comunicaciones de tal forma que, un usuario en cualquier sitio puede acceder a los datos en cualquier parte de la red, exactamente como si los datos estuvieran en un mismo servidor de bases de datos [12].

Los principales factores que distinguen un SBDD de un sistema centralizado son los siguientes:

- Hay múltiples computadores, llamados sitios o nodos.
- Estos sitios deben de estar comunicados por medio de algún tipo de red de comunicaciones para transmitir datos y órdenes entre los nodos.

Las BDD y la replicación de la BD son términos cercanos pero diferentes, esto se evidencia en los siguientes puntos [9]:

- En una BD distribuida pura, el sistema maneja una sola copia de toda la información y soporta los objetos de la BD. Mientras que la replicación confía en tecnología de la BD para funcionar.

- La replicación de la BD puede ofrecer las ventajas de las aplicaciones que no son posibles dentro de un ambiente de BDD puro.

- La replicación es útil para mejorar el funcionamiento y para proteger la disponibilidad de aplicaciones porque existen las opciones alternas del acceso de los datos.

- Además, la aplicación puede continuar funcionando si el servidor local experimenta una falla porque otros servidores con datos replicados siguen siendo accesibles.

1.3.1 Características de la replicación de BD.

Las características principales que debe tener la replicación de BD son [12]:

- **Efectividad:** depende de la forma en la que los datos sean distribuidos y almacenados. A mayor efectividad, mayor será la disponibilidad de datos para ejecutar procesos paralelos.

- **Alta Disponibilidad:** es la razón de tiempo prudente en la que un servicio puede ser accedido. En el mejor de los casos puede ser de un 100%, a pesar de los fallos que se puedan presentar en el servidor, ya que debe existir un servidor adicional que posea alguna técnica de replicación que lo pueda suplantar en caso de ser necesario.

La disponibilidad del sistema se refiere a la accesibilidad de los usuarios a servicios del sistema. Un sistema está disponible si es completamente operacional durante un período de tiempo. La disponibilidad se concentra en la duración del tiempo en el cual se espera que el sistema se mantenga en operación continua o en el caso de recuperación confiable

- **Tolerancia a fallos:** garantiza un comportamiento correcto, donde efectivamente pueden existir un número finito de fallos y tipos de fallos.

- **Coordinación:** cada una de las partes que conforman la base de datos distribuida, acuerda un consenso para realizar las invocaciones de los servicios a los objetos, que al final de la transacción debe realizarse tal y como fue solicitada, para lo cual debe utilizar algún tipo de ordenamiento.

1.3.2 Replicación Básica.

La replicación básica, o simétrica, como también se le conoce, proporciona el acceso de sólo-lectura a los datos de las tablas que provienen de un sitio primario (maestro) [13]. Las aplicaciones pueden preguntar datos de las réplicas de datos locales para evitar el acceso a la red, sin importar su disponibilidad. Sin embargo, las aplicaciones a través del sistema deben tener acceso a los datos en el sitio primario cuando las actualizaciones sean necesarias.

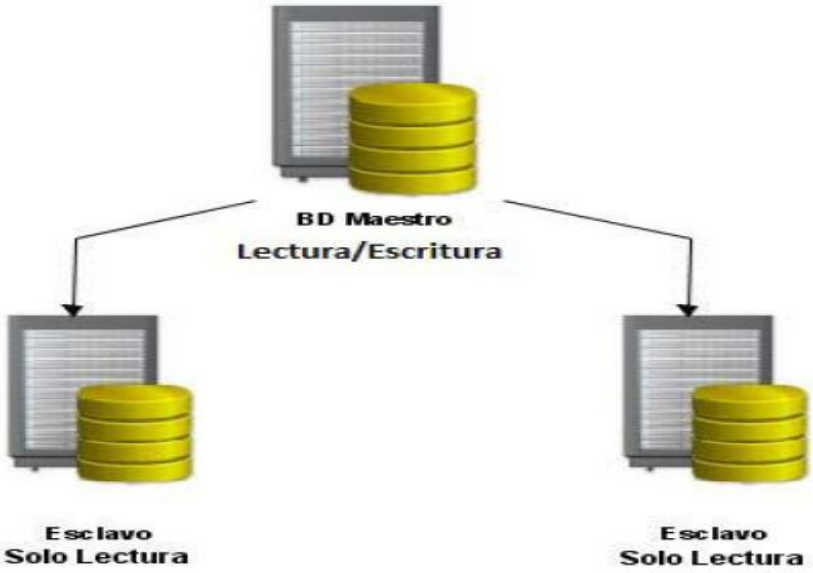


Figura 1: Replicación Básica [9].

Aplicaciones de la Replicación Básica

- **Distribución de la Información:** Este concepto es muy importante para las operaciones que necesiten actualizar datos para garantizar que estén siempre disponibles y actualizados. Esto se traduce en que cada réplica de la base de datos puede tener sus propios datos y estos ser restaurados en la BD.

- **Información Off-Loadin:** La replicación básica es útil como manera de replicar bases de datos enteras o información off-load. Por ejemplo, cuando el funcionamiento de grandes cantidades de transacciones es crítico, puede ser ventajoso mantener una base de datos duplicada para aislar las preguntas exigentes de las aplicaciones de ayuda de decisión.

- **Transporte de la Información:** La replicación básica puede ser útil como mecanismo del transporte de la información. Por ejemplo, la replicación básica puede mover periódicamente datos desde una base de datos de procesamiento transaccional de producción a un almacén de datos.

1.3.3 Replicación (Simétrica) Avanzada

Las características avanzadas de replicación amplían las capacidades básicas de sólo- lectura de la replicación, permitiendo que las aplicaciones hagan actualizaciones a las réplicas de las tablas, a través de un sistema replicado de la base de datos [13]. Con la replicación avanzada, los datos pueden proveer lectura y acceso a actualizaciones a los datos de las tablas.

Aplicaciones de la Replicación Avanzada.

- **Ambientes Desconectados:** La replicación avanzada es útil para el despliegue de las aplicaciones del procesamiento transaccional que funcionan con componentes desconectados. Esto permite que los datos puedan ser enviados a la BD centralizada, aún cuando en el momento en que estos se introdujeron no existía conexión a la red.

- **Sitio de Failover:** La replicación avanzada puede ser útil para proteger la disponibilidad de una base de datos crítica. Por ejemplo, un sistema avanzado de replicación puede replicar una base de datos entera para establecer un sitio de failover, si el sitio primario se convierte en inaccesible debido a un sistema o a una interrupción de la red. Tal sitio del failover puede también servir como base de datos completamente funcional al acceso de la aplicación de ayuda, cuando el sitio primario es concurrentemente operacional.

- **Cargas de aplicaciones distribuidas:** La replicación avanzada es útil para las aplicaciones de tratamiento transaccional, que requieren puntos múltiples de acceso a la información de la base de datos para propósitos de distribuir una carga pesada de aplicación, asegurando disponibilidad continua, o proporcionando más acceso localizados a los datos. Las aplicaciones que tienen tales requisitos comúnmente incluyen servicio de aplicaciones orientadas al cliente.

1.3.4 Elementos de Replicación.

- **Objetos de la Replicación:** Un objeto de replicación es un objeto de la base de datos que existe en los servidores múltiples en un sistema de la base de datos distribuida.

- **Grupos de Replicación:** En un ambiente avanzado, se manejan objetos de replicación usando grupos de replicación. Organizando objetos relacionados de la base de datos dentro de un grupo de replicación, es más fácil administrar muchos objetos juntos. Típicamente, se crea y utiliza un grupo de replicación para organizar el esquema de objetos necesarios para apoyar una aplicación particular de la base de datos. La restricción básica es que un objeto de la replicación puede ser un miembro de solamente un grupo.

- **Sitios de Replicación:** Un grupo de la replicación puede existir en múltiples sitios de replicación. Los ambientes avanzados de replicación soportan dos tipos básicos de sitios: Sitios maestros y sitios de snapshot [1].

[1] Son copias de un sitio que se colocan localmente cerca de otro cliente para mejorar su velocidad de acceso a dicha información, esto se hace solo cuando dicho cliente realiza una gran carga sobre los datos [13].

- **Catálogo de Replicación:** Cada sitio maestro y de snapshot en un ambiente avanzado de replicación tiene un catálogo de replicación. El catálogo de replicación de un sitio es un sistema distinto de diccionario de tablas y vistas que mantienen información administrativa sobre objetos de la replicación y grupos de replicación en el sitio. Cada servidor que participa en un ambiente avanzado de replicación, puede automatizar la replicación de objetos en grupos usando la información en su catálogo de la replicación.

1.3.5 Modelos de Distribución de Datos.

Existen dos modelos de distribución de datos esencialmente aplicados a la replicación de datos [14]:

1. Asincrónica: A menudo llamada *"almacena y reenvía"*, captura cualquier cambio local, los almacena en una cola, y a intervalos regulares, propaga y aplica estos cambios en sitios remotos. Con esta forma de réplica, hay un período de tiempo antes de que todos los sitios alcancen la convergencia de datos.

Algunas técnicas de este modelo de replicación son:

- **Descarga y Recarga (Dump and Reload):** Consiste en hacer un volcado de los datos, copiar la salva para un dispositivo de almacenamiento para luego distribuir la salva por los demás servidores. Esta técnica presenta el inconveniente de que en la mayoría de las ocasiones se consultaban datos que tenían semanas de desactualización, además de que el proceso se realizaba de forma manual.

- **Instantánea (Snapshot):** Consiste en hacer una instantánea de una tabla en un momento dado y esta "imagen" es la que se replica en las demás bases de datos. Aunque mucho más avanzado que la técnica de Descarga y Recarga, esta presenta el inconveniente de que las tablas esclavas son tablas de solo lectura. Se recomienda utilizar cuando: la mayoría de los datos no cambian con frecuencia; se replican pequeñas cantidades de datos; los sitios con frecuencia están desconectados y es aceptable un periodo de latencia largo (la cantidad de tiempo que transcurre entre la actualización de los datos en un sitio y en otro).

- **Disparadores (Triggers):** La función del disparador es ejecutar una acción cuando ocurre un evento en la base de datos, los eventos pueden ser de inserción, actualización o eliminación. Conjuntamente con las instantáneas, los disparadores son otro de los mecanismos asincrónicos que proporciona la base de datos como una manera de replicación de datos.

2. Sincrónica: También conocida como la *"réplica en tiempo real"*, aplica cualquier cambio o ejecuta cualquier procedimiento reproducido en todos los sitios que participan en el ambiente de réplica como parte de una sola transacción. Si el procedimiento falla en cualquier sitio, entonces la transacción entera se anula. La réplica sincrónica asegura la consistencia de datos en todos los sitios en tiempo real. Esta tecnología se llama *Confirmación en dos fases* y surge en los años ´80 [14].

Por lo tanto, la réplica sincrónica de los datos se utiliza solamente cuando las aplicaciones requieren que los sitios replicados sigan sincronizados continuamente.

1.3.6 Entornos de Réplica.

La réplica de datos puede aplicarse de dos maneras diferentes de acuerdo a las necesidades propias de las personas o instituciones que la lleven a cabo, estas son [9]:

- **Maestro-Esclavo:** También es conocido como de *"sólo lectura"*, porque permite a un solo sitio (maestro) realizar consultas de escritura sobre los demás, mientras estos solo pueden hacer consultas de lectura (esclavos). La figura 2 muestra la distribución de este entorno.

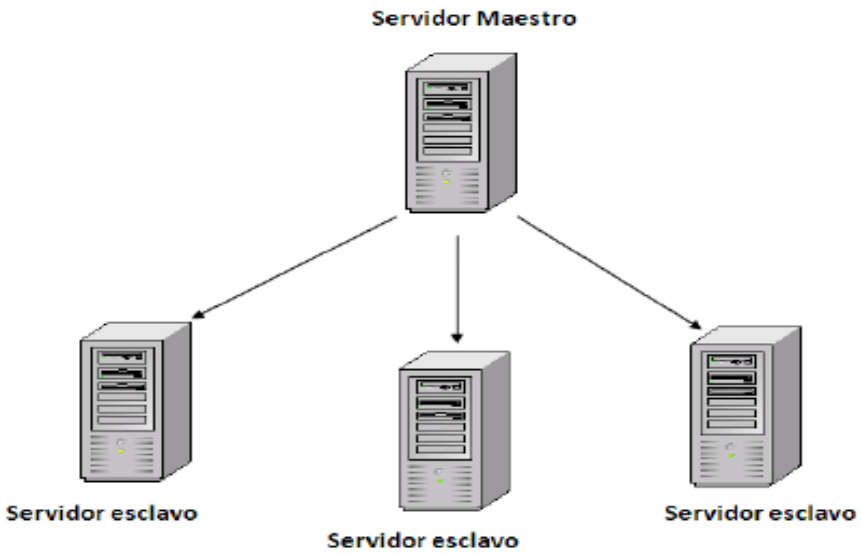


Figura 2: Entorno de Réplica Maestro – Esclavo [12].

- **Maestro-Maestro:** También llamado *"par-a-par o réplica de camino de n"*, porque permite múltiples sitios (maestros), actuando como pares iguales (Figura 3). Cada sitio es un sitio maestro, y se comunica con otros sitios maestros. Esta capacidad tiene también un severo impacto en el desempeño debido a la necesidad de sincronizar los cambios entre todas las partes que intervienen en la réplica.

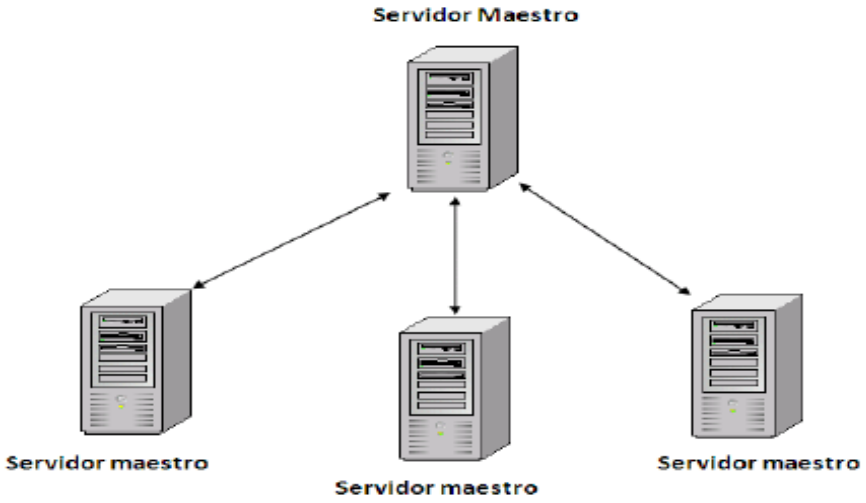


Figura 3: Entorno de Réplica Maestro – Maestro [12].

1.3.7 Ambientes de Replicación.

Un ambiente de replicación es una configuración de dos o más sitios mediante un escenario par a par. Cada sitio es un par que contiene un motor de replicación y una BD simple o compartida. El motor de replicación puede residir en la misma computadora que la BD asociada o en una computadora separada.

En cada caso, la BD debe ser accesible por el cliente del motor de replicación. Cada sitio almacena solo los datos que requieren los usuarios locales. En segundo plano, el motor de replicación gestiona los cambios realizados a la BD sincronizando las actualizaciones de los datos con otros sitios activos en la red.

Una red de replicación puede ser de los siguientes tipos [12]:

- **Homogénea:** Se replican datos entre BD con gestores y plataformas del mismo tipo (ej. PostgreSql + Linux - PostgreSql + Linux).

- **Homogénea con diferentes plataformas:** Se replican datos entre BD con gestores del mismo tipo y plataformas diferentes (ej. PostgreSql + Windows - PostgreSql + Linux).

- **Heterogénea:** Se replican los datos entre bases de datos en gestores diferentes no importa si el sistema operativo sea el mismo o no (ej. PostgreSql + Windows - Oracle + Linux).

Una red de replicación requiere una estructura básica TCP/IP que posibilite una comunicación efectiva y eficiente entre todos los sitios. La red de replicación por sí misma constituye estructuralmente una red virtual que se coloca por encima de la red física.

1.3.8 Conflictos de Replicación de Datos.

Durante la replicación de datos se pueden encontrar un conjunto de conflictos que son de vital importancia su conocimiento. Estos pueden ocurrir cuando se está trabajando en un ambiente de replicación que permite actualizaciones concurrentes sobre los mismos datos en múltiples sitios [15].

1. **Conflicto de actualización**: Ocurre cuando se produce la replicación de una actualización sobre un registro con otra actualización sobre el mismo registro, es decir, cuando dos transacciones originadas desde distintos sitios actualizan el mismo registro, en forma cercana en el tiempo.

   Resolución:

   - **Prioridad**: Cada servidor obtiene una prioridad única, y el servidor de mayor prioridad "gana", respecto de aquellos con prioridad menor.
   - **Timestamp:** La más nueva o la más antigua de las modificaciones es la considerada correcta, y por defecto, si no se eligió ninguno de los criterios "gana" la más nueva.
   - **Particionamiento de datos**: Se garantiza que cada registro sea manipulado por un único servidor, lo que simplifica la arquitectura.

2. **Conflicto de unicidad**: El conflicto de unicidad sucede cuando la replicación de un registro intenta violar una restricción de integridad, ya sea por llave primaria o única, es decir, cuando dos transacciones originadas de dos sitios diferentes, insertan un registro en su respectiva tabla replicada, con el mismo valor de clave primaria.

   Resolución:

   - Para cada servidor brindar un rango distinto de números para los generadores de clave (secuencias).
   - Agregar el identificador del servidor a la clave primaria.
   - Replicar en tablas separadas, y acceder a los datos a través de una vista formada por la unión de ellas. Para resolver el conflicto de potenciales claves duplicadas en la unión se usará una pseudo columna que representa la base de datos fuente.

3. **Conflicto de supresión**: Un conflicto de supresión ocurre cuando dos transacciones originadas de sitios diferentes, una de ellas intenta borrar un registro, y la otra actualizar o borrar el mismo registro, que en este caso el registro no existe, tanto para ser actualizado como borrado.

   Resolución:

   - Para evitar este tipo de conflictos, una posible solución es que los sitios marquen lógicamente los registros a ser borrados y que periódicamente el sitio maestro corra un proceso que realice el borrado físico de los datos, es decir desde los sitios replicados no se puede ejecutar una sentencia para hacer el borrado de los datos.

4. **Conflicto de orden**: Los conflictos de orden pueden ocurrir en ambientes de replicación con tres o más sitios maestros. Si la propagación al sitio maestro X, está bloqueada por alguna razón, entonces la replicación de modificaciones en datos puede seguir siendo propagada a través de los otros sitios maestros. Al finalizar la propagación, estas modificaciones debieron ser propagadas al sitio X en un orden diferente a como ocurrieron en los otros sitios maestros, pudiendo producirse un conflicto.

   Resolución:

   - Este tipo de conflictos suelen resolverse asignándole distintas prioridades a los sitios maestros, de manera que se puedan ordenar las transacciones de acuerdo a esta.

1.4 Herramientas de réplica sobre PostgreSql.

1.4.1 Bucardo multi-master.

Bucardo es un sistema de replicación maestro - maestro y maestro – esclavo para PostgreSql que utiliza desencadenadores (término en inglés: triggers) en tablas individuales. Se encarga de la resolución de conflictos y gestión de excepciones mediante el uso de subrutinas de Perl personalizadas [16].

Requerimientos necesarios para el uso de Bucardo multi-master.

1. Gestor de Base de Datos PostgreSql.

Bucardo utiliza a PostgreSql como gestor de bases de datos, cuya versión necesaria debe ser de 8.1 o superior y debe tener instalado además, el lenguaje Pl/Pgsql[1]. La base de datos que Bucardo utiliza debe tener instalado el lenguaje Pl/Perl[2].

2. Lenguaje de Programación Perl

Bucardo se ejecuta como una serie de demonios de Perl (versión 5.8.3 o superior), es decir en segundo plano. Algunos de los módulos que se requieren para su ejecución son:

- **DBI 1.51:** Permite conectar a varias bases de datos de distintos tipos y mover fácilmente datos entre ellas. Tim Bunce desarrollador de este módulo desde 1994 lo define como un módulo estándar de interfaz de base de datos para Perl, que define un conjunto de métodos, variables y convenciones proporcionando una interfaz de base de datos consistente independiente de la base de datos real que se utiliza [19].

- **DBD::Pg 2.0:** Funciona con el módulo DBI para proporcionar acceso a las bases de datos PostgreSql [20].

- **DBIx::Safe 1.2.4:** Realiza versiones más seguras de las bases de datos que maneja DBI limitando qué métodos se pueden realizar y cuáles se permiten hacer [21].

- **Moose 0.18:** Es una extensión que brinda modernas características al lenguaje orientado a objetos de Perl, haciendo la programación más coherente y menos tediosa. Moose es construida en el tope de la clase MOP, que es un sistema de metaclases para Perl 5 que proporciona no solo la construcción mejor de objetos, sino que mantiene el poder de la programación metaclase, es decir, las clases cuyas instancias son clases [21].

- **IO::Handle 1.24:** Es utilizado para pasar los objetos normales de Perl a funciones [22].

- **Sys::Hostname 1.11:** Realiza la función de conseguir los nombres de host en la red comenzando por averiguar el host que está llamando su anfitrión, almacenando el resultado en caché [23].

3. Sistema operativo tipo Unix

[1] PL/Pgsql: Procedural Language/PostgreSQL Structured Query Language es un lenguaje imperativo que permite ejecutar comandos SQL mediante el uso de funciones dando mucho más control automático que las sentencias SQL básicas [17].

[2] Pl/Perl: Es un lenguaje procedimental ejecutable que permite escribir las funciones de PostgreSQL en el lenguaje Perl [18].

Bucardo se ha probado en sistemas operativos (SO) Linux y BSD (sistemas operativos derivados de UNIX), pero no se puede ejecutar en el SO Windows sin pequeñas modificaciones en el código que participa en las llamadas al sistema [15].

Limitaciones de Bucardo.

- Se requiere tener instalado la versión 8.1 de PostgreSql o superior, con los lenguajes Pl/Perl y Pl/Pgsql versión 8.2 o superior.
- Se requiere la última versión de Perl y DBD::Pg.
- Solamente replica las tablas, no es así con una base de datos completa.
- No se puede manejar dos nodos maestros a la vez, es decir la replicación máster-máster-máster no la incluye todavía.
- Requiere de una llave primaria en cada tabla para que pueda ser replicada.
- Al no hacer seguimiento continuo en los cambios de datos en aquellas redes que son inestables en la conexión las copias de seguridad a realizar son muy costosas.

1.4.2 Slony.

Slony es un sistema de replicación “maestro a múltiples esclavos”, y es implementado por la Comunidad de PostgreSql. Dentro de sus características se encuentra el soporte a la réplica en cascada y permite a un esclavo ser a su vez maestro para otro servidor (Figura 4). Incluye los tipos de funcionalidades necesarias para replicar bases de datos grandes a un número razonablemente limitado de sistemas esclavos. En este contexto razonable es un número no superior a 10 servidores. Si el número de servidores crece más allá de 10, el costo de las comunicaciones aumentará prohibitivamente, y las ventajas incrementales de tener servidores múltiples fallarán en ese punto.

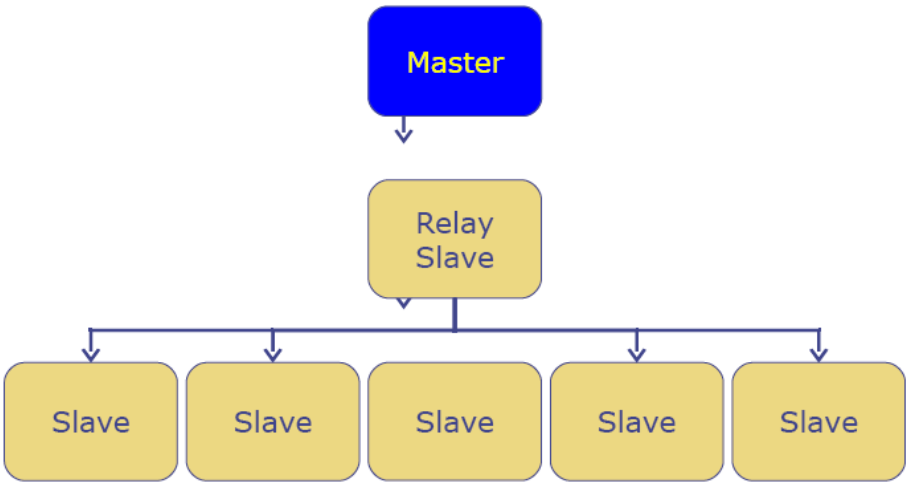


Figura 4: Muestra de la replicación a múltiples esclavos, siendo un esclavo a su vez maestro para otros esclavos [23].

Slony-I implementa la réplica asincrónica, 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. 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 [24].

1.4.3 Pyreplica.

Pyreplica es una herramienta multiplataforma que permite la replicación asincrónica de datos para PostgreSql, utilizando disparadores implementados en pl/python, señales, secuencias y un script cliente influenciado por Slony pero mucho más simple y fácil. Soporta además la replicación en un entorno maestro-maestro, donde se replican datos en ambas direcciones [25].

Está programado en Python por lo que es simple y flexible, permitiendo una fácil instalación. El usuario no necesita aprender nuevos comandos para su administración, porque es muy fácil de adaptar manualmente. Este mecanismo es muy eficiente en el uso de memoria y la red, manteniendo un bajo impacto en estas al no utilizar transmisión secuencial (polling).

Otra de las características de Pyreplica es la detección de conflictos de actualización/eliminación y falla en conflictos de inserción o errores de integridad de datos. Sin embargo no permite la replicación de cambios de esquema, es decir, los comandos CREATE, ALTER, etc, deben ejecutarse manualmente en todos los servidores [25].

Pyreplica soporta además, todos los tipos de datos soportados por Pl/Python que pueden ser representados como cadenas (incluyendo bytea).

Algunas mediciones de rendimiento simple que se le han realizado a este mecanismo, muestran que el disparador solo es un 50% más lento que uno basado en C (como slony-I), con los beneficios de que puede ser fácilmente instalado, portado, mantenido y adaptado.

1.4.4 DBReplicator.

DBReplicator es una poderosa aplicación de código abierto escrita en Java para la replicación multi-maestro de bases de datos heterogéneas. Soporta varios gestores de bases de datos tales como: FireBird, Oracle, PostgreSql, MySQL, SQLServer, entre otras.

Para trabajar con el gestor PostgreSql es necesario configurar los ficheros pg_hba.conf y postgresql.conf para habilitar las conexiones TCP en direcciones de red local.

Las principales características de DBReplicator son [26]:

- **Replicación Heterogénea:** Como se mencionó anteriormente, DBReplicator puede realizar la replicación heterogénea, es decir la sincronización de una misma base de datos entre múltiples Sistemas de Gestores de Bases de Datos Relacionales (Siglas en inglés: RDBMS, Relational DataBase Management System).

- **Sincronización bi-direccional de datos** entre cualquier sistema de control de base de datos soportado.

- **Detección y resolución automática de conflictos:** Los administradores de bases de datos requieren una participación mínima en la detección y resolución de conflictos ya que muchas de las versiones de DBReplicator poseen algoritmos para realizar esta tarea.

- **Facilidad de programación:** El sistema puede programarse para realizar operaciones en distintos intervalos de tiempo que pueden ser entre horas, minutos o segundos.

- **Creación de tablas:** Crea automáticamente las tablas que no existen en las bases de datos subscritas para que coincidan con las bases de datos originales.

Algunas versiones de DBReplicator incluyen el uso de banderas para indicar el estado de replicación de los registros de una tabla. Estos estados son: N = no replicado, P = pendiente y S = sincronizado. También algunas de ellas manejan automáticamente los campos UTF8[1] o caracteres Unicode[2].

1.4.5 SymmetricDS.

SymmetricDS es un sistema de software libre, escrito en Java, liberado bajo los términos de la LGPL (Lesser General Public License: Licencia Pública de Linux). Este software se instala como una aplicación web dentro de un servidor de aplicaciones Java, o puede ser incorporado a otra aplicación Java.

Es un software de replicación de datos asíncrona que permite subscriptores múltiples y sincronización bidireccional. Utiliza tecnologías web y de bases de datos para replicar tablas entre bases de datos relacionales, casi en tiempo real. El software fue diseñado para escalar a un gran número de bases de datos, trabajar con conexiones de bajo ancho de banda, y resistir a períodos de inoperatividad de la red.

SymmetricDS soporta la sincronización entre diferentes plataformas de base de datos (MySQL, Oracle, SQL Server, PostgreSql, DB2, Firebird, HSQLDB y H2), mediante el concepto de dialectos de base de datos [27].

Un dialecto de base de datos es una capa de abstracción con la cual interactúa SymmetricDS para aislar la lógica de sincronización de los detalles de implementación específicos de cada base de datos [28].

Una de las ventajas que presenta SymmetricDS es la sincronización bidireccional de datos, la cual se define a nivel de tablas, evitando incurrir en bucles de actualización porque sólo registra cambios de datos fuera de la sincronización, es decir, le asigna canales a cada flujo, de manera que las actualizaciones se hagan de forma independiente.

Las instalaciones SymmetricDS van creando Nodos (Figura 5). Un Nodo es inicializado mediante un fichero “properties”, y es configurado insertando datos de configuración en una serie de tablas de base de datos. A continuación, el Nodo crea triggers de base de datos en las tablas de aplicación especificadas, de modo que los eventos de base de datos son capturados para ser entregados a otros Nodos SymmetricDS. De esta manera se realiza el proceso de réplica.

[1] Formato de codificación de caracteres Unicode.
[2] Estándar de codificación de caracteres para facilitar el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y disciplinas técnicas [7].

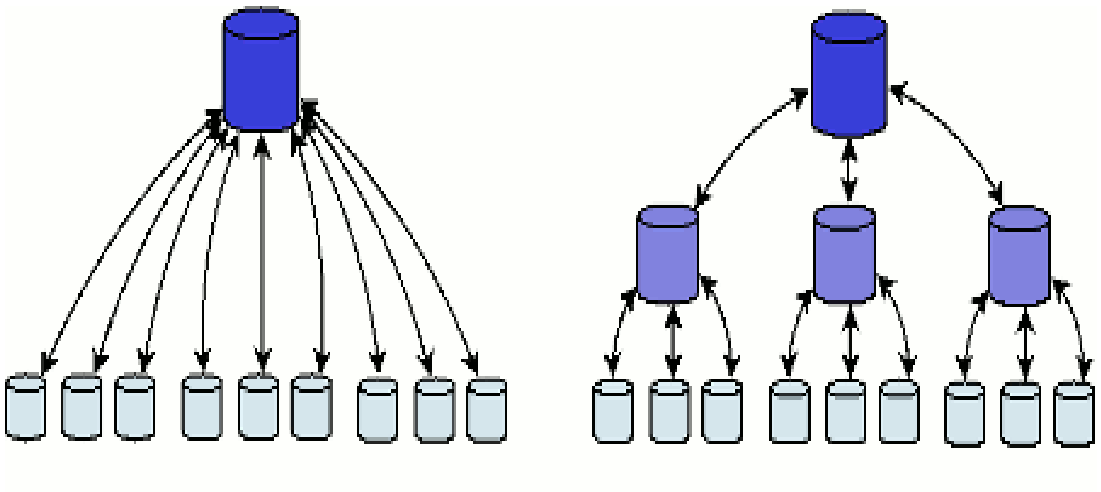


Figura 5: Modo de réplica de SymmetricDS. Creación de Nodos [27].

1.5 Clúster de Bases de Datos.

En el presente epígrafe se aborda el tema de los clúster de BD que no son más que un conjunto de ordenadores que se comportan como si fuera una sola computadora con el objetivo de lograr un alto rendimiento, alta disponibilidad, escalabilidad así como una alta eficiencia, todas estas características en su conjunto hacen que sea mucho más potente que una computadora de escritorio.

1.5.1 PgCluster.

Es un sistema de replicación sincrónico multi-maestro para PostgreSql 7.4.6 o superior. Consiste en tres tipos de servidores: un servidor para balance de carga, un clúster de base de datos y un servidor de réplica. Este mecanismo posee dos funciones principales:

- **Compartir carga:** La carga de la sesión de las demandas es distribuida. Es efectivo en aplicaciones web donde existe gran demanda por el número de peticiones.

- **Alta disponibilidad:** Cuando ocurre un fallo en el Clúster BD, el servidor de balance de carga y el de replicación separan el fallo del sistema, y continúa el servicio que usa el BD restante. El Clúster BD cuando es reparado puede restaurarse dinámicamente a un sistema, sin detener el servicio. Los datos son copiados automáticamente a la BD restaurada o añadidos desde otra BD.

Desde el punto de vista de la réplica multi-maestro el PgCluster envía todas las consultas desde el servidor de réplica hacia cada clúster de base de datos (BD), luego el servidor de réplicas bloquea

otras peticiones a través de un semáforo hasta que las transacciones hayan sido terminadas. De esta misma forma sucede para cada transacción manteniendo el orden de la misma manera en que hayan sido recibidas [29].

1.5.2 CyberCluster.

Este mecanismo es una variante del PgCluster mucho más eficiente porque mantiene todos los nodos (clúster) de las bases de datos en sincronización todo el tiempo. Esto tiene como ventaja que cuando un nodo falla, el clúster automáticamente no solamente lo detecta, sino que resuelve el problema, a diferencia del PgCluster [30].

El CyberCluster es un sistema de código abierto y está liberado bajo los términos de la licencia BSD[1], es ideal para aplicaciones de alto rendimiento y puede correr sobre varios sistemas operativos como Windows, Linux (en gran variedad de distribuciones), Mac OS X, etc. Es el primer software a escala empresarial que proporciona tecnología de replicación de alto rendimiento a las compañías que utilizan PostgreSql.

Cybercluster tiene soporte en otras características que son muy importantes a la hora de diseñar e implementar un sistema de réplica puro, entre ellas se encuentran [11]:

- **DDL[2]:** Soportado.
- **Protocolo de confirmación en dos fases:** Soportado.
- **Restricciones:** Al reproducir un objeto grande, necesita ser puesto en un directorio que pueda ser leído por todos los nodos del banco de datos.
- **Racimo de los nodos de la DB:** Los Clústeres (nodos del racimo) son las máquinas que procesan las demandas reales que entran. Siempre que una demanda se reciba del cliente un Clúster de BD tiene que determinar si está enfrentando una demanda de lectura o escritura. La demanda de lectura es ejecutada directamente por el banco de datos y el resultado se envía al cliente. En caso de que sea una demanda de escritura, el Clúster de BD envía un mensaje al administrador de la replicación para asegurarse que ese dato se envía a todos los nodos del banco de datos restantes dentro del racimo.

[1] BSD (Berkeley Software Distribution): Es una licencia de software libre permisiva que permite el uso del código fuente en software no libre [7].
[2] DDL- Data Definition Language, Lenguaje de definición de datos.

- **Servidor de la Replicación:** El servidor de la Replicación es el componente central del sistema, que toma demandas entrantes solo de los nodos del banco de datos y copia esos cambios a todos los nodos del banco de datos en el sistema. Si el servidor de la replicación descubre un problema en un nodo del banco de datos, quitará la máquina de la lista de banco de datos activos y se escribirá al disco un log que contiene una descripción detallada del error. Esto ocurre para asegurar que los nodos del banco de datos nunca puedan estar fuera de sincronización en caso de fallo. Basado en los logs[1] el administrador del sistema puede decidir entonces si es posible agregar la máquina de nuevo al racimo.

Servidor de Balanceador de Carga: Cybercluster contiene su propio balanceador de carga, que asegura que todos los nodos de los bancos de datos pueden trabajar eficazmente al mismo tiempo. En la práctica esto significa que el acceso de lectura puede ser casi infinito. Este balanceador se usa para distribuir la carga dentro del racimo. La carga en el sistema es determinada por el número de preguntas activas. La máquina con el número más bajo de preguntas activas será escogida para realizar una nueva demanda.

Si el balanceador de carga detecta un problema en un nodo activo del banco de datos automáticamente destaca este nodo dentro del sistema activo para asegurarse que ninguna otra solicitud se enviará a esta máquina rota.

El Balanceador de Carga usado con Cybercluster es opcional. Si ningún balanceador de carga está disponible, las aplicaciones se pueden conectar a cualquier nodo del banco de datos dentro del sistema. El propio balanceador de carga por sí mismo no está de ninguna manera involucrado en la replicación, simplemente se usa para distribuir la carga [11].

1.5.3 Pgpool.

Pgpool es un programa intermediario (middleware) realizado con código C, que trabaja entre PostgreSql y servidores de una base de datos cliente. Dentro de sus características se encuentran el ser un balanceador de carga, además de poseer una fácil instalación, configuración, código estable y flexible. Otros elementos importantes de este mecanismo son [31]:

- Trabaja entre los servidores de datos y el cliente para reducir el consumo (overhead) de establecimiento de la conexión.

- Trabaja hasta con 2 servidores, si el principal cae automáticamente trabaja con el otro.

[1] Registros de Bitácora.

- Puesta en común de conexión, guarda conexiones con el servidor PostgreSql, y la reutilización siempre que sea una nueva relación con las mismas propiedades (es decir, nombre de usuario, bases de datos, versión del protocolo). Reduce de conexión y mejora el rendimiento general del sistema.

- Permite la replicación ya que puede gestionar múltiples servidores PostgreSql. El uso de la función de replicación en tiempo real permite crear una copia de seguridad en 2 o más discos físicos, por lo que el servicio pueda continuar sin parar los servidores en caso de un fallo de disco.

- Limita conexiones superiores: hay un límite en el número máximo de conexiones simultáneas con PostgreSql, y las conexiones se rechazan después de esta cantidad de conexiones.

Conclusiones Parciales

- El uso de bases de datos en el mundo ha venido extendiéndose con el incremento cada vez mayor de los volúmenes de datos, cuyo manejo se realiza a través de Sistemas Gestores de Bases de Datos.

- Algunos gestores de bases de datos incluyen mecanismos de réplicas potentes muy utilizados a nivel mundial, aunque no todos son software libre.

- Los proyectos de desarrollo en el país abogan por la utilización de PostgreSql como gestor de bases de datos, a pesar de que sus soluciones de réplica no son las más seguras y presentan problemas cuando se trata de aplicaciones a gran escala que se extienden a través de redes grandes como las WAN.

- Se hace necesario proponer una solución de réplica robusta que permita garantizar la integridad de los datos en la ONRM a través de redes WAN.

Capítulo 2: Propuesta de solución.

En el presente capítulo se plantea la propuesta de SymmetricDS como una solución de replicación de datos eficiente para PostgreSql en la ONRM. Se detallan las características principales y la arquitectura de esta herramienta, de manera que sea mucho más fácil su comprensión. Como parte de los objetivos propuestos para esta fase de la investigación, se elabora una guía para la configuración e instalación de SymmetricDS, mostrando para ello los pasos necesarios para su utilización.

2.1 SymmetricDS como herramienta de replicación.

En el capítulo anterior se realizó de manera general un resumen de las características de SymmetricDS como una de las herramientas de replicación para PostgreSql. En este epígrafe se propone realizar una caracterización detallada de esta herramienta, atendiendo a los aspectos específicos que van a ser útiles para comprender su funcionamiento.

2.1.1 Principales Características.

Dentro de las características de SymmetricDS se encuentran [28]:

- **Notificación de Esquemas.**

Tras registrar un cambio que se ha producido en la base de datos, los nodos interesados en el cambio son notificados. La notificación de cambios se configura para realizar un push[1] o un pull[2]. Cuando varios nodos dirigen sus cambios a un nodo central, es eficiente realizar un push de los cambios, en vez de esperar a que el nodo central realice un poll de cada nodo origen. Cuando la configuración de red protege a un nodo con un firewall, una configuración pull permite que el nodo reciba cambios de datos que de otro modo podrían ser bloqueados utilizando push. La frecuencia de la notificación de cambios es de un minuto en la configuración por defecto.

- **Canales de Datos.**

La sincronización de datos se define al nivel de tabla (o de subconjunto de tablas). Cada tabla gestionada puede ser asignada a un canal que ayuda a controlar el flujo de datos. Un canal es una categoría de datos que puede ser habilitada, priorizada y sincronizada independientemente del resto de los canales.

[1] Push: Replicar datos [31].
[2] Pull: Recuperar datos [31].

- **Transacción de integridad.**

Muchas bases de datos proporcionan un identificador de transacción asociado a las filas que son actualizadas (commited) conjuntamente. SymmetricDS almacena el ID de transacción junto con los datos que han cambiado, de modo que puede volver a ejecutar la transacción exactamente del mismo modo que ocurrió. Esto significa que la base de datos de destino mantiene la misma integridad que la fuente.

- **Filtrado y re-enrutado de Datos.**

El filtrado y enrutado de datos es especificado utilizando expresiones SQL para crear disparadores (triggers) que son instalados por SymmetricDS para captar los cambios de datos.

SymmetricDS posee tres clases de filtrado:

- ✓ **IDataLoaderFiltered:** Puede cambiar los datos en una columna o enrutarlos a otro lugar. Esto es muy útil cuando se quiere proteger datos de contraseñas y para prevenir que los datos lleguen a la base de datos en conjunto, reemplazando la carga de datos predeterminada.
- ✓ **IColumnFilter:** Permite excluir columnas de la sincronización de manera que estas no cambien cuando la tabla lo hace.
- ✓ **IExtractorListener:** Utilizada para filtrar los datos que son extraídos de la base de datos central y enrutarlos hacia otro lugar.
- ✓ **Transporte HTTP.**

SymmetricDS posee un sistema basado en web que muestra una Representación del Estado de la Transferencia de Datos, el cual posee una serie de filtros para restringir el número de flujos de sincronización simultáneos.

- **Administración Remota.**

Esta función es posible a través de Java Management Extensions (JMX)[1], que pueden ser accedidas desde la consola de Java o a través de un servidor de aplicaciones. Las operaciones que se pueden realizar son: *apertura de registros*, *carga de datos*, *depuración de datos caducados*, *etc*. SymmetricDS provee además una funcionalidad para enviar eventos SQL a través del propio mecanismo de sincronización que utiliza para enviar datos.

[1] JMX: Es una tecnología que define una arquitectura de gestión para la administración de aplicaciones Java.

2.1.2 Requerimientos.

SymmetricDS fue programado utilizando la tecnología Java, por lo que para su ejecución es necesario poseer una máquina virtual (versión 5.0 o superior), pues Java se compila a través de un código intermedio que es interpretado por la JVM (Java Virtual Machine) que también se le conoce como JRE (Java Runtime Environment).

Siempre y cuando una base de datos posea tecnología de disparo (triggers) y un controlador JDBC puede soportar réplica con SymmetricDS. Para atender las características específicas de cada base de datos se utilizan los dialectos de bases de datos y entre ellos se encuentran:

- MySQL versión 5.0.2 o superior.
- Oracle versión 8.1.7 o superior.
- PostgreSql versión 8.2.5 o superior.
- SQLServer 2005.
- HSQLDB 1.8.
- H2 1.1.
- Apache Derby 10.3.2.1 o superior.
- FireBird 2.0 o superior.

2.2 Conceptos Principales.

Nodo de base de datos: Es una base de datos distribuida que tiene la posibilidad de manejar sus propios datos [32].

JDBC (Java Database Conectivity): Se utiliza para conectar un programa de usuario con una base de datos en segundo plano, sin importar qué software de administración o manejo de base de datos se utilice para controlarlo [33].

HSQLDB (Hyperthreaded Structured Query Language Database): Es un sistema gestor de bases de datos libre escrito en Java. La suite ofimática OpenOffice.org lo incluye desde su versión 2.0 para dar soporte a la aplicación Base [34].

H2: Sistema administrador de bases de datos relacionales programado en Java. Puede ser incorporado en aplicaciones Java o ejecutarse de modo cliente-servidor. Una de las características más importantes de H2 es que se puede integrar completamente en aplicaciones Java y acceder a la base de datos lanzando sentencias SQL directamente [35].

Apache Derby: Sistema gestor de base de datos relacional escrito en Java que puede ser embebido en aplicaciones Java y utilizado para procesos de transacciones online. Tiene un tamaño de 2 MB de espacio en disco. Apache Derby es un proyecto open source licenciado bajo la Apache 2.0 License. Actualmente se distribuye como Sun Java DB [36].

FireBird: Sistema de administración de base de datos relacional multiplataforma y de código abierto con muy buena seguridad basada en usuarios/roles. Dentro de sus características importantes se encuentra que ofrece una concurrencia excelente, alto rendimiento y un poderoso lenguaje de procedimientos almacenados y disparadores [37].

Spring Framework: También conocido como Spring, es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. A pesar de que Spring Framework no obliga a usar un modelo de programación en particular, se ha popularizado en la comunidad de programadores en Java al considerársele una alternativa y sustituto del modelo de Enterprise JavaBean. Por su diseño, el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria. Una de las características esenciales de Spring es que posee una capa de abstracción común para la gestión de transacciones y una para JDBC [38].

Jetty: Servidor HTTP y un contenedor de Servlets escrito en Java. Jetty se publica como un proyecto de software libre bajo la licencia Apache 2.0. Debido a su pequeño tamaño, Jetty se complementa para ofrecer servicios Web en una aplicación Java empotrada [39].

2.3 Arquitectura.

La arquitectura de SymmetricDS está configurada de manera que un nodo pueda actuar tanto como cliente o como host. El cliente es quien inicia la conexión y el host es el receptor. El hecho de comportarse de una u otra forma es debido a la sincronización bidireccional de SymmetricDS.

El software está compuesto por una serie de manejadores, servlets y servicios conectados entre si, haciendo uso de Spring Framework, el cual permite el manejo de dependencias.

El proceso de conexión de un cliente con un host (anfitrión) comienza cuando se activa el *PushJob* y el *PullJob* para sincronizar con un nodo anfitrión (Ver Figura 6). Para el trabajo de inyección (*PushJob*) se necesitan utilizar los servicios de empuje (*PushService*), que permiten extraer (*CsvExtractor*) y establecer el flujo de datos con otro nodo. Por su parte el *PullJob* utiliza los servicios de carga de datos (*DataLoaderService* y *CsvLoader*) que se transmiten desde otro nodo. Cuando se establece la comunicación entonces se realiza una segunda conexión para enviar confirmaciones de que esta fue realizada o no.

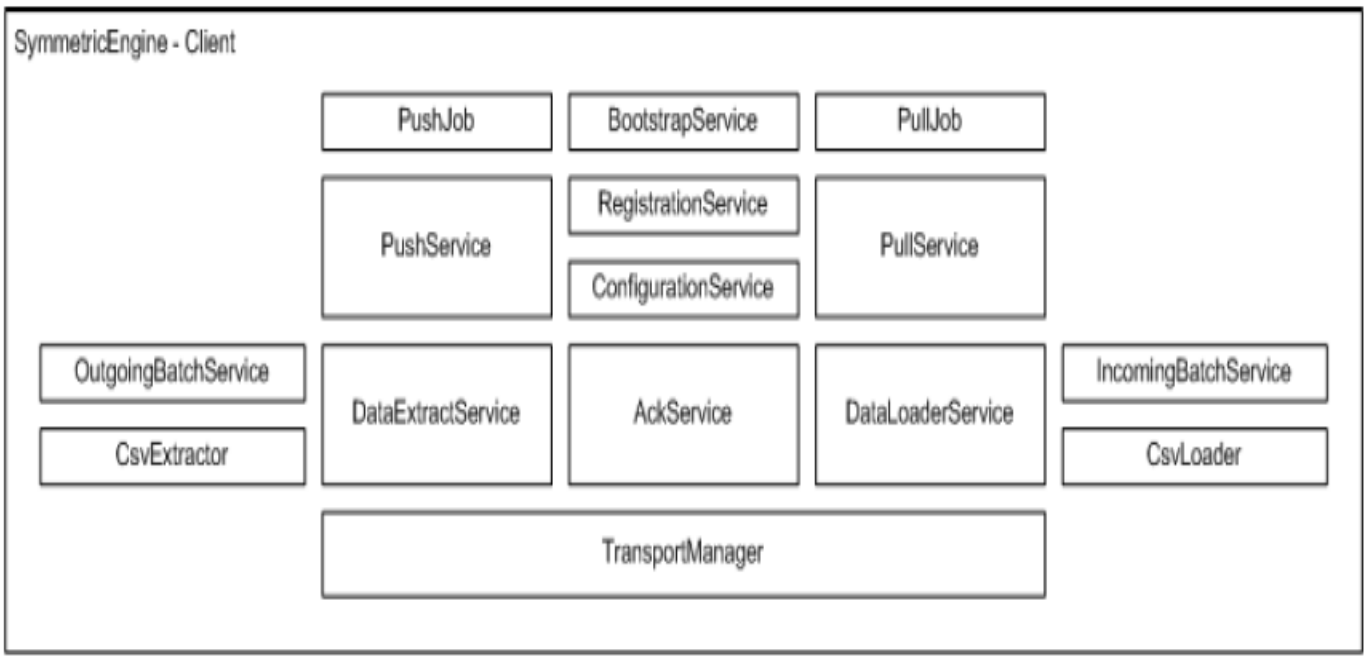


Figura 6: Arquitectura de Symmetric como cliente [28].

SymmetricDS como host realiza la espera por las conexiones ya sean de recuperación, replicación o notificaciones de cambios realizadas por un cliente. A través del *Pushservlet* el host utiliza los servicios de carga de datos (*DataLoaderService*) que son inyectados por un cliente (Ver Figura 7) y luego se envían las confirmaciones para comprobar que las operaciones se hayan realizado o no. A través del *PullServlet* se utilizan los servicios de extraer y establecer flujos de datos devueltos al cliente. El *AckServlet* actualiza el estado de los datos que son cargados en un nodo cliente.

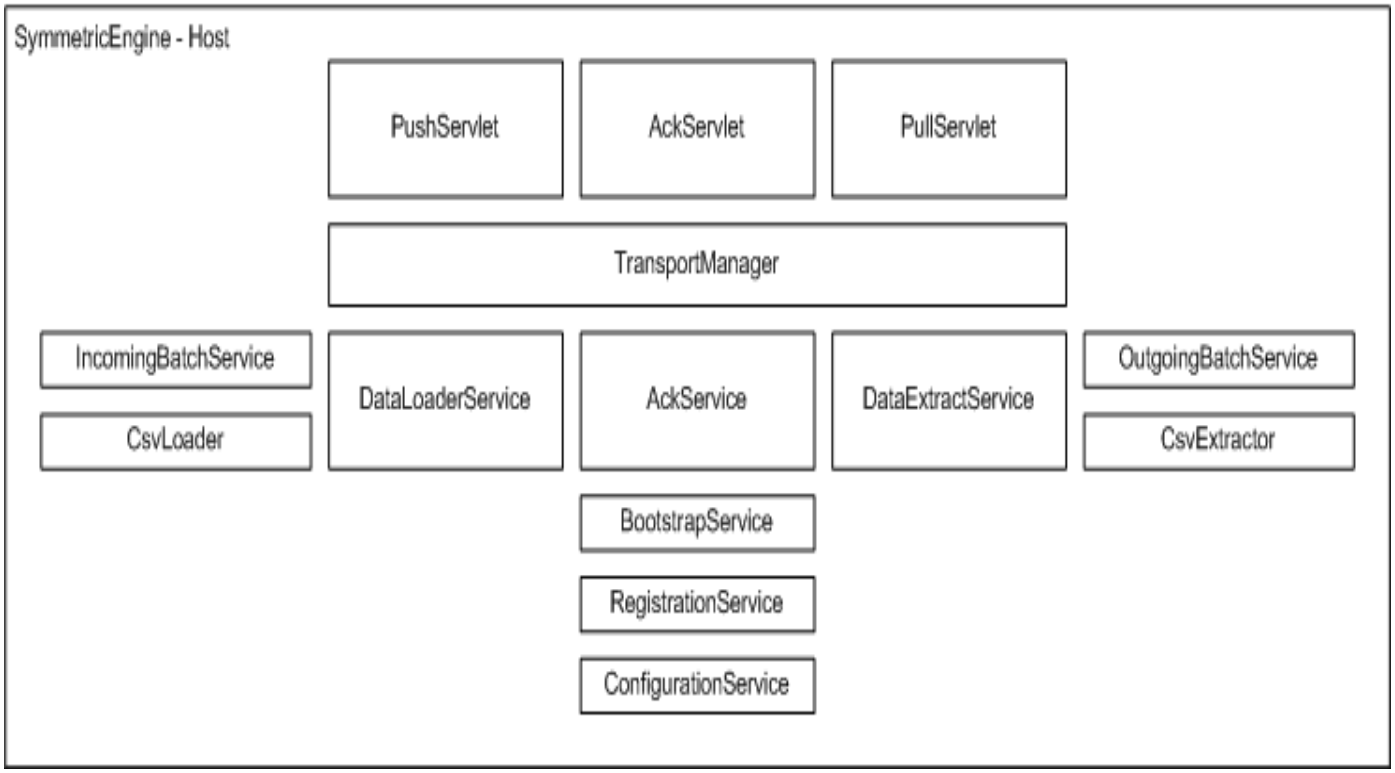


Figura 7: Arquitectura de Symmetric como host [28].

El *TransportManager* es el encargado de manejar los datos de entrada y salida entre los nodos. Esta operación se realiza por defecto vía http y el flujo de eventos para la comunicación ente los nodos se muestra en la figura 8.

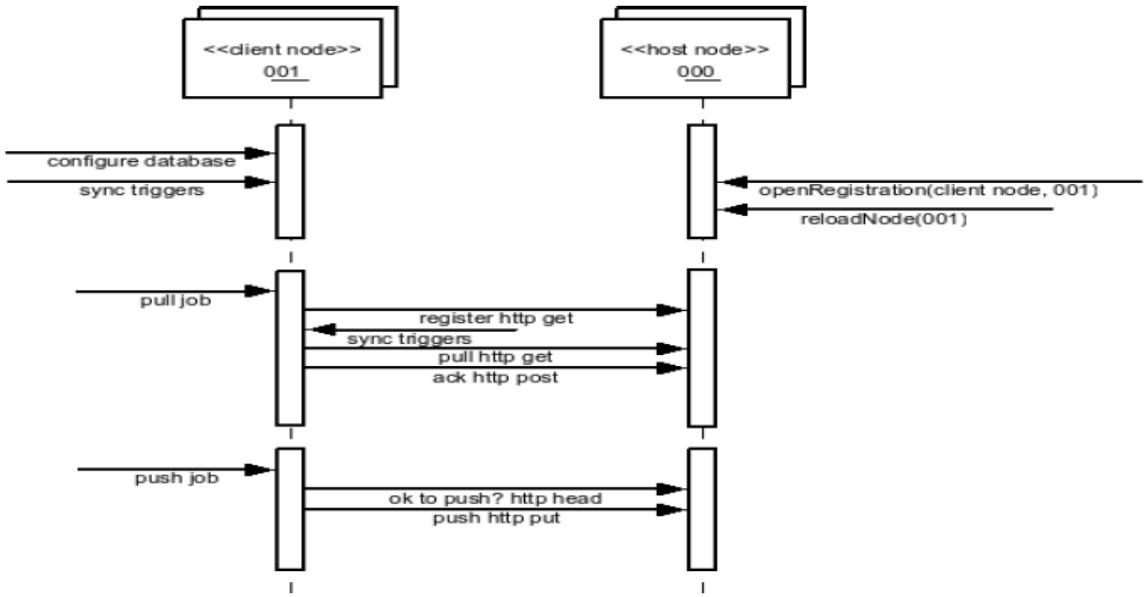


Figura 8: Flujo del proceso de comunicación entre nodos [28].

2.4 Estructura de SymmetricDS.

SymmetricDS está compuesto por:

- Un archivo de aplicación Web (Figura 9), desplegado en un servidor de aplicaciones.

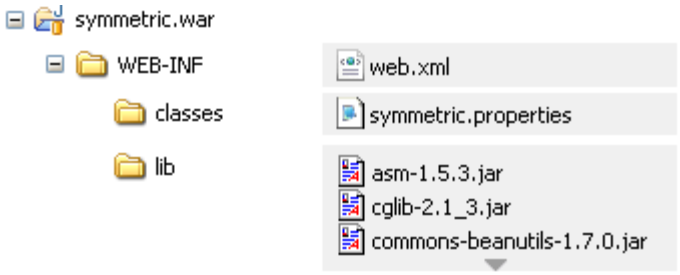


Figura 9: Vista de los archivos de configuración de la aplicación Web de SymmetricDS [28].

- Un servicio independiente que embebe un servidor web Jetty.
- Un sistema de librerías Java.

2.4.1 La aplicación Web.

La figura que se presenta a continuación, muestra el contenido del archivo *WEB-INF/web.xml* el cual es configurado con el *SymmetricEngineContextLoaderListener*, el *SymmetrisFilter* y el *SymmetricServlet* (archivos de configuración).

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   version="2.5">
   <display-name>sync</display-name>
   <context-param>
      <param-name>contextConfigLocation</param-name>
      <!-- You can optionally specify other Spring files to load into same context here --
      <param-value>classpath:symmetric.xml</param-value>
   </context-param>
   <filter>
      <filter-name>SymmetricFilter</filter-name>
      <filter-class>
         org.jumpmind.symmetric.web.SymmetricFilter
      </filter-class>
   </filter>
   <filter-mapping>
      <filter-name>SymmetricFilter</filter-name>
      <servlet-name>/*</servlet-name>
   </filter-mapping>
   <listener>
      <listener-class>
         org.jumpmind.symmetric.SymmetricEngineContextLoaderListener
      </listener-class>
```

```
  </listener>
  <servlet>
    <servlet-name>SymmetricServlet</servlet-name>
    <servlet-class>
      org.jumpmind.symmetric.web.SymmetricServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>SymmetricServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Figura 10: Vista del archivo web.xml [28].

En este ejemplo se inician todos los Servlets de SymmetricDS para comprimir el flujo, los nodos de autenticación, y rechazar los nodos cuando el servidor está muy ocupado. Se pueden establecer propiedades en el archivo *web.base.servlet.path* en *symmetric.properties* si *SymmetricServlet* necesita coexistir con otros servlets.

2.4.2 El servicio independiente.

Este servicio puede usar la línea de comando (Figura 11) de SymmetricDS para iniciar el servidor. La instancia embebida de Jetty es utilizada para atender solicitudes web de todos los servlets.

```
/symmetric/bin/sym --properties root.properties --port 8080 --server
```

Figura 11: Ejemplo de como iniciar el servidor desde la línea de comandos del SymmetricDS [28].

2.4.3 El sistema de librerías java.

La aplicación Java en conjunto con la librería (jar) Java de SymmetricDS (Figura 12) puede utilizar el SymmetrcEnfine para iniciar el servidor.

```
import org.jumpmind.symmetric.SymmetricEngine;

public class StartSymmetricDSEngine {

   public static void main(String[] args) throws Exception {

      String workingDirectory = System.getProperty("user.dir");

      SymmetricWebServer node = new SymmetricWebServer(new SymmetricEngine("classpath://my

            + workingDirectory + "/my-environment.properties"));

      // this will create the database, sync triggers, start jobs running
```

```
        node.start(8080);
        // this will stop the node
        node.stop();
    }
}
```

Figura 12: Vista del Archivo (jar) Java de SymmetricDS [28].

En el ejemplo de la Figura 11 se inicia el servidor de SymmetricDS sobre el puerto 8080. El primer archivo, *my-application.properties*, se empaqueta en la solicitud para proporcionar propiedades que sobrescriben los valores por defecto de SymmetricDS. El segundo archivo, *my-environment.properties*, se encuentra en un directorio de trabajo que sobrescribe propiedades específicas del ambiente de SymmetricDS. Esto permite que la misma solicitud para implementar el ambiente sea usada para implementar varios ambientes de producción.

2.5 Guía de instalación de SymmetricDS.

En este epígrafe se presentan los pasos para instalar SymmetricDS con todos sus componentes.

2.5.1 Instalando SymmetricDS.

1. Instale el software SymmetricDS y configúrelo con la información de su base de datos de conexión. Descargar el SymmetricDS-1.6.X.zip archivo de http://www.symmetricds.org/.

2. Descomprimir el archivo en cualquier directorio que se elija. Esto creará un subdirectorio de SymmetricDS-1.6.x, lo que corresponde a la versión que ha descargado.

3. Editar las propiedades de la base de datos en los siguientes archivos de propiedades para el directorio raíz y los nodos clientes:

- Muestras / root.properties.
- Muestras / client.properties.

4. Establezca las siguientes propiedades en ambos archivos para especificar como conectarse a la base de datos:

```
# The class name for the JDBC Driver
```

```
db.driver=com.mysql.jdbc.Driver

# The JDBC URL used to connect to the database

db.url=jdbc:mysql://localhost/sample

# The user to login as who can create and update tables

db.user=symmetric

# The password for the user to login as

db.password=secret
```

Figura 13: Estableciendo propiedades para conectarse a la base de datos.

5. Establezca la siguiente propiedad (Figura 13) en el archivo client.properties para especificar que el nodo raíz puede ser contactado:

```
# The HTTP URL of the root node to contact for registration
registration.url=http://localhost:8080/sync
```

Figura 14: Estableciendo propiedades del nodo raíz.

En este ejemplo, la base de datos cliente comienza vacía y el nodo no está registrado. El Registro es el proceso donde el nodo recibe su configuración y almacena sus datos. La configuración describe qué tablas se van a sincronizar y con cuáles nodos. Cuando un nodo no registrado se inicia, se registra con el nodo especificado por la dirección de registro.

2.5.2 Creando y llenando las bases de datos.

Primero: se deben crear las bases de datos raíz y los nodos clientes mediante la administración de herramientas proporcionadas por el proveedor de base de datos. Asegúrese de que el nombre de las bases de datos a crear coincide con la configuración en los archivos de propiedades. Cree las tablas de ejemplo en el nodo raíz de la base de datos, cargue los datos de la muestra, y la configuración de ejemplo.

1. Abra una terminal de comandos y navegue hasta el subdirectorio *samples* de la instalación de SymmetricDS.

2. Cree las tablas de ejemplo en la base de datos raíz:

```
.. / bin / sim-root.properties p - Run-create_sample.xml ddl
```

3. Cree las tablas de SymmetricDS en el nodo raíz de la base de datos. Estas tablas contendrán la configuración de la sincronización. El siguiente comando se utiliza para crear todas las tablas que SymmetricDS necesita:

```
../bin/sym -p root.properties --auto-create
```

4. Cargue los datos de ejemplo y la configuración en el nodo raíz de la base de datos:

```
../bin/sym -p root.properties --run-sql insert_sample.sql
```

Segundo: se deben crear las tablas de ejemplo en el nodo cliente de la base de datos para prepararlo para recibir datos.

1. Abra una terminal de comando y navegue hasta el subdirectorio *samples* de la instalación de SymmetricDS.

2. Cree las tablas de ejemplo en la base de datos cliente:

```
.. / bin / sim-client.properties p - Run-create_sample.xml
```

Tercero: Verifique ambas bases de datos en el registro y el listado de las tablas.

1. Encuentre las tablas que se sincronizan desde la raíz al cliente: ítem e ítem_selling_price.

2. Encuentre los pesos de las tablas que se sincronizan desde el cliente a la raíz: sale_transaction y sale_return_line_item.

3. Encuentre las tablas de SymmetricDS las cuales tienen el prefijo “sym_”.

2.5.3 Iniciando SymmetricDS.

Inicie los nodos de SymmetricDS y observe el registro de salida.

1. Abra una terminal de comandos y navegue hasta el subdirectorio samples de SymmetricDS.

2. Inicie el servidor del nodo raíz, el cual provoca la inicialización de todos los triggers que fueron configurados en el archivo samples.

```
../bin/sym -p root.properties --port 8080 –server
```

3. Inicie el servidor del nodo cliente, el cual es encargado de crear todas las tablas de SymmetricDS. Se inicia entonces el flujo en el orden de registro.

```
.. / bin / sim-client.properties p - puerto 9090 – servidor
```

Se aconseja cambiar el puerto utilizado por SymmetricDS y cambiar la propiedad de tiempo de ejecución my.url, cuyo valor por defecto es:

```
my.url=http://localhost:8080/sync
```

2.5.4 Registrando un nodo.

Abra la inscripción del nodo cliente usando la administración del nodo raíz.

1. Abra una terminal de comandos y navegue hasta el subdirectorio *samples* de SymmetricDS.

2. Abra la inscripción del nodo cliente, por ejemplo:

```
../bin/sym -p root.properties --open-registration "store,1"
```

En este ejemplo el registro se abre para un grupo de nodos llamados *store*, cuyo identificador es 1. Esta información coincide con la configuración del nodo cliente en *client.properties*. Cada nodo es asignado a un grupo y se le brinda un *id* externo. Esto es parte de la Transacción de Awareness, abordada en el epígrafe 2.1.

3. Verifique el registro de salida del nodo cliente para comprobar que se registra correctamente en el nodo raíz. El nodo cliente está configurado para que corra a cada minuto para comprobar que exista conexión y una vez conectados ambos (cliente y raíz) están habilitados para la sincronización.

2.5.5 Enviando la carga inicial.

Envíe una carga inicial del nodo cliente usando la administración del nodo raíz.

1. Abra una terminal de comandos y navegue hasta el subdirectorio *samples* de SymmetricDS.

2. Enviar una carga inicial de datos al servidor del nodo cliente.

```
../bin/sym -p root.properties --reload-node 1
```

Con este comando las colas del nodo raíz levantan una carga inicial de datos para el nodo cliente, la cual será enviada la próxima vez que este último realice un pull. La carga inicial incluye datos para cada tabla configurada para la sincronización.

3. Observe el registro de salida de ambos nodos para ver la transferencia de datos. El nodo cliente está configurado para hacer un pull para el nodo raíz a cada minuto.

2.5.6 Transferencia de datos (pulling).

Modifique los datos de la base de datos raíz, para que los cambios sean propagados al cliente durante la transferencia sincronizada.

1. Abra una sesión interactiva SQL con la base de datos raíz.

2. Añada un nuevo punto de venta para este ejemplo:

```
insert into item_selling_price (price_id, price) values (55, 0.65);

insert into item (item_id, price_id, name) values (110000055, 55, 'Soft Drink');
```

Una vez que los datos son actualizados, entonces se activa la cola para comenzar la transferencia hacia el nodo cliente.

3. Observe el registro de salida de ambos nodos para ver la transferencia de datos. El nodo cliente está configurado para hacer un pull para el nodo raíz a cada minuto.

4. Verifique que los nuevos datos lleguen al cliente usando una nueva sesión interactiva SQL.

2.5.7 Avance de los datos (pushing).

Modificar los datos de la base de datos cliente, para que los cambios sean propagados a la raíz durante el avance de forma sincronizada.

1. Abra una sesión interactiva SQL con la base de datos cliente.

2. Añada un nuevo punto de venta para este ejemplo:

```
insert into sale_transaction (tran_id, store, workstation, day, seq) values (1000, '1', '3', '2007-11-01', 100);

insert into sale_return_line_item (tran_id, item_id, price, quantity) values (1000, 110000055, 0.65, 1);
```

Una vez que los datos son actualizados, entonces se activa la cola para comenzar la transferencia hacia el nodo raíz.

3. Observe el registro de salida de ambos nodos para ver la transferencia de datos. El nodo cliente está configurado para hacer un pull para el nodo raíz a cada minuto.

2.5.8 Verificando las salidas.

En este caso se utilizan los batch, los cuales son empleados para el seguimiento y el envío de los cambios de datos a los nodos. El nodo emisor genera un batch, el cual es identificado por el nodo receptor durante la transferencia.

1. Abra una sesión interactiva SQL, ya sea con el nodo raíz o cliente.

2. Compruebe que fue capturado el cambio de datos:

%% sale el tema SELECT * FROM nombre_tabla donde sym_data como "" o como nombre_tabla ';

Cada fila representa una fila de datos que se ha cambiado. Para insertar y actualizar, los datos son listados en *row_data* y para eliminar y actualizar, lo son en *pk_data*.

3. Compruebe que el cambio de los datos fue enviado a un nodo, utilizando el data_id del paso anterior:

SELECT * FROM sym_data_event donde data_id =?;

Cuando se establece la bandera batch, el cambio de datos está asignado a un batch con un *batch_id* que se utiliza para seguir y sincronizar los datos. Los batches son creados y asignados durante un pull y un push en la sincronización.

4. Compruebe que el batch de cambio de datos fue enviado y reconocido con el batch_id del paso anterior:

SELECT * FROM sym_outgoing_batch donde batch_id =?;

Un batch representa un conjunto de cambios para ser enviados a un nodo. El batch se forma cuando se hace un pull o un push en la sincronización y cuando el estado se define como "NE" para los nuevos. El nodo receptor reconoce el batch con un estado de "OK" para el éxito o "ER" para el error.

5. Compruebe que se ha registrado la historia de batches, utilizando el batch_id del paso anterior anterior:

SELECT * FROM sym_outgoing_batch_hist donde batch_id =?;

El trabajo realizado en el batch se registra en la historia de la tabla. Un nuevo grupo con la condición de "NE" registra el número de cambios de datos que contiene en el campo *data_event_count*. La condición de "SE" muestra que un batch se envía a un nodo. El estado de acuse de recibo desde el

nodo de recepción también se registra. Si el estado es error, el *failed_data_id* indica que la fila en *sym_data* ha sido error.

2.5.9 Verificando los batches entrantes.

El nodo receptor realiza un seguimiento de los batches que reconoce y registra las estadísticas sobre la carga de los datos. Los que son duplicados se omiten por defecto, pero esto puede ser cambiado con la propiedad *incoming.batches.skip.duplicates*.

1. Abra una sesión interactiva SQL, ya sea con el nodo raíz o cliente.

2. Compruebe que se reconoció el batch, con un batch_id de la sección anterior:

```
SELECT * FROM sym_incoming_batch donde batch_id =?;
```

3. Compruebe que se ha registrado la historia de batches, utilizando el batch_id de la etapa anterior:

```
SELECT * FROM sym_incoming_batch_hist donde batch_id =?;
```

El trabajo realizado en el batch se registra en la historia de la tabla. Si un batch duplicado se ha omitido, el estado se registra como "SK". De lo contrario, la situación es "OK" para el éxito o "ER" para el error. El *statement_count* es el número de filas y la carga *byte_count* es el tamaño de los batches en bytes. El *database_millis* es la cantidad de tiempo en milisegundos empleados en cargar los datos en la base de datos.

Conclusiones Parciales

- SymmetricDS es una herramienta de replicación robusta, que soporta la arquitectura Cliente - Servidor.
- Dentro de sus características principales se incluyen la *Notificación de Esquemas*, los *Canales de Datos*, *la Transacción de integridad*, el *Filtrado y re-enrutado de Datos* y la *Administración Remota.*

- La arquitectura de SymmetricDS está configurada de manera que un nodo pueda actuar tanto como cliente o como host.

- Su estructura se compone de una *aplicación web*, un *servicio independiente* y un *sistema de librerías*.

- Para instalar y configurar SymmetricDS se deben llenar las bases de datos que van a actuar como cliente o como host. Luego del inicio del sistema, se procede al registro de un nodo cliente o raíz y para comprobar que ambos nodos están listos para la sincronización, se envía una carga inicial de datos. Se observa el pulling y el pushing de datos y las salidas que estos producen.

Capítulo 3: Aplicación de SymmetricDS como solución de réplica en la Oficina Nacional de Recursos Minerales.

En este capítulo se presenta la propuesta de solución, exponiendo de forma organizada y detallada cuál será el procedimiento a seguir para dar respuesta a la situación problémica planteada. Se describen los recursos que posee la ONRM, así como las herramientas y tecnologías utilizadas en el entorno de desarrollo. Se exponen los requerimientos necesarios para la implantación de la solución de réplica y se explica además cómo quedará establecido el despliegue de la solución de acuerdo a la descripción realizada.

3.1 Descripción del entorno.

El presente epígrafe realiza una descripción del entorno de desarrollo de la ONRM desde el punto de vista arquitectónico, reflejando para ello las tecnologías y herramientas utilizadas en el mismo.

La nueva solución propone llevar a los usuarios de la ONRM y el MINBAS la utilización de una solución eficaz y eficiente que cumpla con los objetivos estratégicos de la organización, garantizando con su implantación, la centralización de los recursos de información, así como la disminución del costo de mantenimiento, alta tolerancia a fallas y un alto nivel de accesibilidad para los usuarios finales.

La información de la ONRM es accesible solamente desde la red interna de la corporación y el servidor de mapas que se encuentra en el MINBAS es de acceso a todos, tanto desde la red local así como desde Internet. Debido a estas políticas de acceso y de acuerdo a la confidencialidad de la información publicada en la ONRM estos portales se encuentran físicamente separados, en lo referente a redes y a los servidores evitando así que un ataque informático al portal en Internet pueda dañar las aplicaciones de la empresa o acceder la información interna.
La Intranet de la ONRM, cuya nueva versión es una aplicación web implementada sobre el CMS Drupal, es el medio en el cual se publican los contenidos noticiosos que se redactan en dicha institución y están dirigidos al personal de la entidad. Es además el Entorno de Trabajo Colaborativo Virtual de la ONRM y el punto de acceso a los servicios que componen la plataforma tecnológica, es decir, a través de la Intranet los usuarios podrán acceder a toda la información y los servicios necesarios para desempeñar su rol en la empresa. Tales razones convierten a la Intranet en un ambiente de alta concurrencia por lo que es de vital importancia garantizar bajos tiempos de respuesta y una alta disponibilidad en los servicios que presta.

El servidor de mapas en Internet, por su parte, es una puerta informativa que trata de llevar al público externo la realidad de la ONRM a través de la publicación de los contenidos generados en el MINBAS. Teniendo en cuenta el público potencial, en este caso toda persona con acceso a Internet, debe ser tratado como un sistema de alta concurrencia. Para la implementación de la nueva versión, el proyecto propone el uso del CMS Drupal en la capa de aplicaciones.

Si se analiza el esquema de la solución propuesta se puede apreciar que los datos necesarios para satisfacer las solicitudes a los servicios de noticias que generan los usuarios internos y externos, se encuentran almacenados en las bases de datos de la ONRM. Esto trae consigo que la concurrencia en la BD de la ONRM dependa de la suma de peticiones a los servicios de noticias de los medios digitales mencionados.

El alto número de concurrencias a las BD y la complejidad de cada una de las transacciones pueden provocar la interrupción de los servicios de un servidor de BD, causado por el gran consumo de procesamiento o memoria.

3.2 Descripción de la infraestructura de la ONRM.

Los servicios que posee actualmente la institución están distribuidos en tres servidores de bases de datos, el servidor de aplicaciones, el servidor de mapas y un servidor de BD montado en Ubuntu de marca DEL- PowerEdge 2900 situado en la ONRM que es donde se encuentra toda la información referente a dicha entidad con un Microprocesador Intel Core 2 Duo cuya velocidad es de 2.2 Ghz, posee una Memoria RAM de 2 GB con dos discos duros de 73 GB, la comunicación entre dichos servidores es por el protocolo TCP/IP a una velocidad de 128 KB/s y están interconectados en una red LAN. Por otra parte en el MINBAS existen otros dos servidores de BD que también están en una red LAN y se comunican por el protocolo TCP/IP éstos son, el servidor de Mapas y otro servidor con las BD de las aplicaciones que están publicadas en Internet que se llama servidor Portal que tiene las siguientes características: memoria RAM de 512 MB, posee además un disco duro de 160 GB y un Microprocesador con una velocidad 2.66 GB.

A continuación se describen las características que debe cumplir la propuesta de esta solución.

I. La réplica debe realizarse en un solo sentido.

II. La solución tendría que ser capaz de funcionar aunque existan interrupciones en la red.

III. La información debe permanecer disponible y actualizada en cada uno de los nodos.

IV. Los servidores contarían con sistema operativo GNU/Linux.

V. El nodo central debe contener la totalidad de la información de cada una de las oficinas o filiales asociadas.

VI. En el servidor central y en los nodos asociados a él debe contar con alguno de los siguientes SGBD como MySQL, Oracle, SQL Server, PostgreSql, DB2, Firebird, HSQLDB, H2, y Apache Derby

3.3 Propuesta de despliegue de solución.

Después de haber comprendido cómo está constituido el entorno productivo de la ONRM, está todo listo para empezar a describir cómo se realizará el despliegue de solución en el mismo. Para ello se deberá entender el funcionamiento integrado de los servicios que se encuentran en dicho centro. La figura 15 muestra una vista de estas aplicaciones.

Como punto de partida se explica que la replicación se hará de manera unidireccional, es decir, que los datos van a migrar solamente desde la ONRM hacia el MINBAS en un solo sentido de dirección para ello se necesitan varias máquinas con sistema operativo Linux, una sería para el servidor central y al menos dos nodos clientes de manera que se comporte como un sistema distribuido con un servidor central, además se debe tener instalado la versión 8.4 de PostgreSql o superior, el PgAdmin III, Máquina Virtual de Java y el driver JDBC para la conexión a la BD. La herramienta cuenta con la característica que puede replicar los datos ya sea en una red LAN mediante el protocolo TCP/IP o en redes WAN mediante el protocolo http, ya sea de una forma u otra, no debe ser preocupación para el cliente la cantidad de información que se quiera replicar por muy grande que sea ni por el formato que tenga pues SymmetricDS divide esa información en pequeños paquetes y replica sin problemas a una velocidad mínima de 64 Kb/s, que en el caso de la ONRM como es de 128 Kb/s no tendrá problemas de carga en la red ni de conexión con los servidores. El tipo de replicación que se utilizará en la ONRM será maestro a esclavo como se muestra en la figura 16 porque permite a un solo sitio maestro (en este caso la ONRM) realizar consultas de escritura sobre los demás, mientras éstos solo pueden hacer consultas de lectura (esclavo, en este caso el MINBAS).

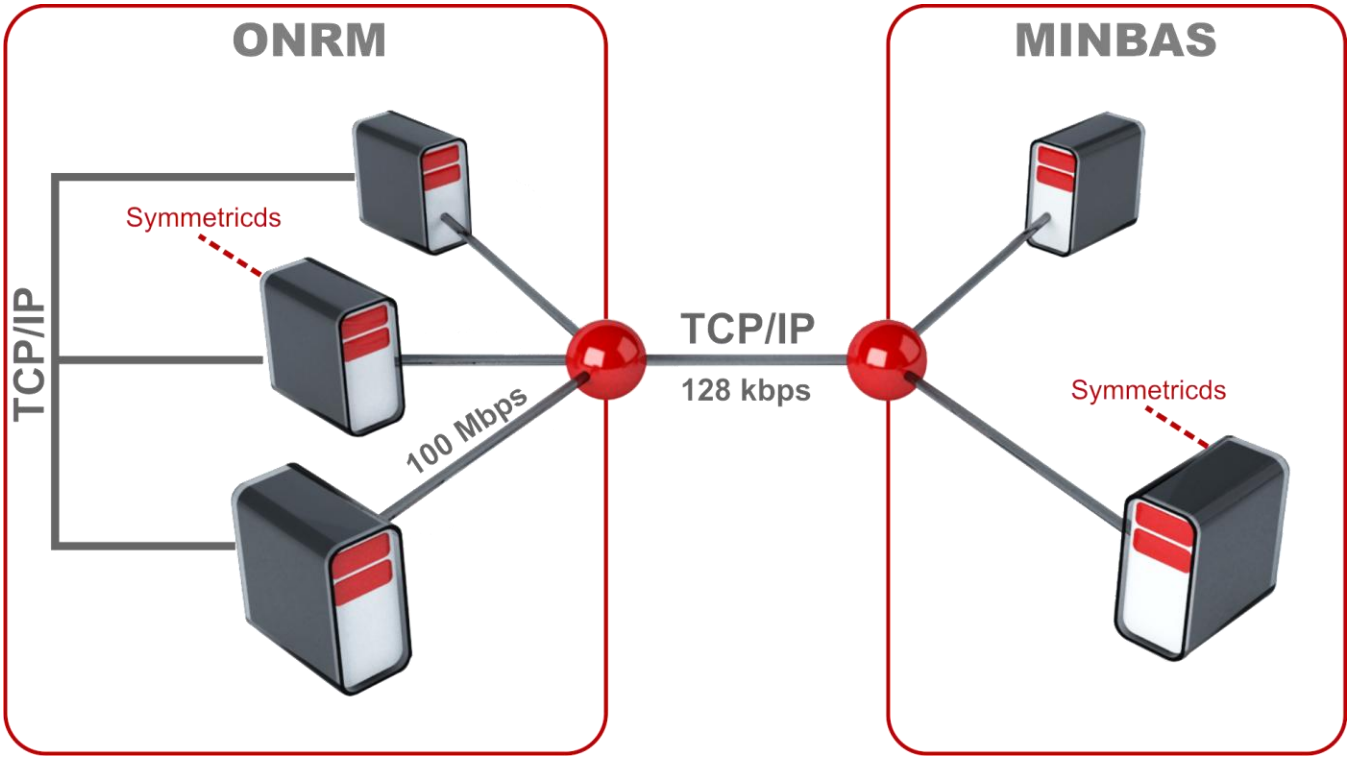


Figura 15: Vista de las aplicaciones.

El horario que se propone para replicar los datos es bien temprano en la mañana, de 7:00 AM a 8:00 AM o en el horario de almuerzo de 12:00 PM a 1:00 PM, para evitar que estén muchos usuarios conectados a las aplicaciones y así evitar sobrecarga en la red de la ONRM.

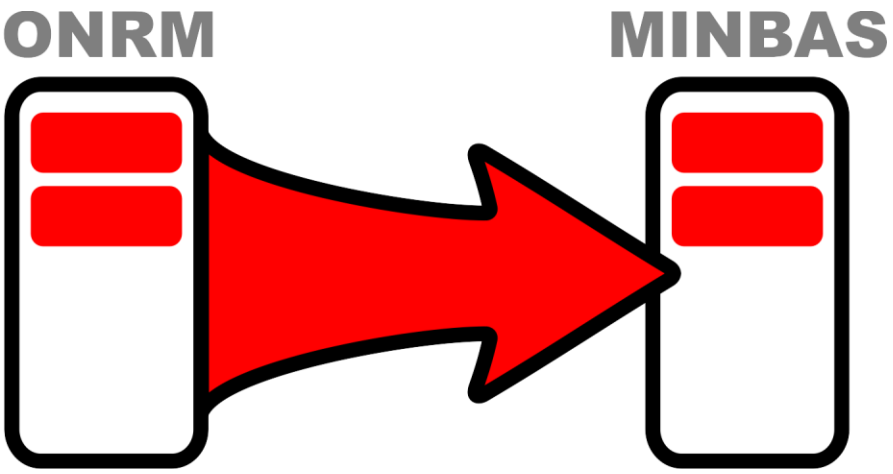


Figura 16: Réplica unidireccional.

3.3.1 Descripción del entorno de prueba.

Una vez explicado el despliegue es hora de describir en todo ese entorno donde van a estar situadas físicamente las instancias de SymmetricDS, para ello básicamente se necesita saber qué BD se quiere replicar, es decir, su estructura, la descripción de las tablas con sus campos y tipos de datos, además que información dentro de esa BD se desea replicar de un lado a otro. En la figura 17 se muestran las instancias de SymmetricDS en el entorno de réplica final.

Se implantó la herramienta en el entorno real de la ONRM para ello se replicó la BD SGDG y la BD Portal, a continuación se describen detalladamente los 10 esquemas que tiene la BD SGDG con sus respectivas tablas: balancea tiene 11 tablas, balancem 42 tablas, balancep 18 tablas, bdreferativa 15 tablas, común 2 tablas, concesionariom 39 tablas, concesionariop 18 tablas, metadatos 54 tablas, nomencladores 8 tablas y seguridad tiene 4 tablas, mientras que Portal cuenta con 4 esquemas, común presenta 2 tablas, concesionariom 1 tabla, public 74 tablas y directorio tiene 2 tablas.

3.3.2 Pruebas realizadas.

Para validar el correcto funcionamiento de la solución propuesta es necesario el montaje de un entorno de desarrollo similar al definido anteriormente, al cual se le realizarán pruebas, las cuales proporcionan datos que permiten obtener una visión del futuro comportamiento del sistema. En principio se realizaron 3 pruebas: Prueba de Funcionamiento, Prueba de Conectividad y Prueba de Integridad.

3.3.2.1 Prueba de Funcionamiento.

Objetivo: Es un tipo de prueba con el objetivo de determinar la respuesta, el rendimiento, la confiabilidad y la escalabilidad de un sistema bajo una carga de trabajo determinada.

Escenario: Para la ejecución de las pruebas se utilizó un entorno similar al de la ONRM. A partir de la propuesta de solución definida se decide aplicar un entorno de réplica único en el cual estarán todas las tablas de las BD que se quieren replicar así como la información y la descripción de las mismas. El sistema de réplica está compuesto por 2 servidores HP Proliant BL 460c G1 con las siguientes características:

- Memoria RAM: 1Gb
- CPU: Core 2 Duo 2.00 GHz
- Sistema Operativo: Ubuntu 9.10 karmic 32 bits
- Configuración de Red: LAN 100Mb

En este tipo de prueba se seleccionó del esquema balancea la tabla tparametrocomunes donde se insertaron los siguientes datos:

Insert into balancea.tparametrocomunes (idparametrocomunes, mineralización, ph, temperatura, fecha)

Values (56, 56, 55, 55,2010-06-23);

Se realizaron las siguientes operaciones de actualización:

Update balancea.tparametroscomunes set idparametrocomunes=556, mineralización=57, ph=56, temperatura=56, fecha=2010-06-23, where<idparametrocomunes=557>;

Se realizaron las siguientes operaciones de eliminación:

Delete from balancea.tparametrocomunes where idparametrocomunes==557

Se realizaron pruebas con mayores volúmenes de datos, aproximadamente 1 mg de datos, en el caso de la inserción se probó sobre las tablas del esquema balancea mostradas anteriormente con un tiempo de réplica aproximado de 5 segundos, el tiempo que demora la actualización es aproximadamente 5 segundos y el tiempo que se demora en el proceso de eliminación es de aproximadamente 5 segundos, obteniéndose todos los datos en la BD destino.

Resultados: Al realizar la confirmación de escritura en el nodo esclavo se verificó que los datos fueron insertados íntegramente en la BD destino, los datos replicaron en determinadas tuplas en la tabla escogida ya sea al insertar, eliminar o actualizar datos.

3.3.2.2 Prueba de Conectividad.

Objetivo: Es un tipo de prueba con el objetivo de determinar la respuesta, el rendimiento, la confiabilidad y la escalabilidad de un sistema para cuando exista problemas de conectividad.

Escenario: Para la ejecución de esta prueba se desconecta la red del servidor donde está situada la BD destino y se continúa insertando datos en el nodo maestro hasta poblar completamente la misma con datos válidos.

Resultados: Con la aplicación de esta prueba se demostró que se insertaron datos en la base de datos origen y al desconectar la red el mismo continuaba insertando datos de modo que al restablecer la conexión se verificó que esos datos fueron replicados garantizando la integridad de los mismos.

3.3.2.3 Prueba de Integridad.

Objetivo: La integridad de una base de datos significa que, la base de datos ó los programas que generaron su contenido, incorporen métodos que aseguren que el contenido de los datos del sistema no se corrompa así como las reglas del negocio. Tiene como objetivo asegurar que los métodos de acceso y los procesos funcionen apropiadamente y sin corrupción de datos.

Las pruebas de integridad de datos se realizan invocando métodos de acceso a la BD, intentando con datos válidos e inválidos. También es necesario inspeccionar la BD para asegurar que ha sido poblada como se esperaba, que todos los eventos ocurran apropiadamente, o revisar los datos retornados para asegurar que se obtuvieron los datos correctos.

Criterio de cumplimiento
Todos los métodos de acceso a la base de datos y procesos funcionan como fueron diseñados y sin corrupción de datos. Para el caso de las BDD se hace necesario verificar que los datos sean consistentes en las BD de cada nodo.

Escenario: Para la ejecución de esta prueba se utilizó como entorno los dos servidores descritos anteriormente, se insertaron datos en la BD origen hasta poblar completamente la BD destino. Se insertaron datos de tipo integer, float, double, etc desde la BD origen y se probó que llegan íntegramente a la BD destino sin modificar el tipo de dato.

Resultados: Los datos que se insertaron en la BD origen se actualizan en la BD destino en un tiempo aproximado de 3 segundos, asegurando la confiabilidad y la integridad de los mismos. Se verificó que el sistema es capaz de manejar los diferentes tipos de datos (integer, float, etc) utilizados en las tablas de las bases de datos de las aplicaciones de la ONRM.

3.3.2.4 Análisis de los resultados.

De los resultados obtenidos se concluye que para las pruebas realizadas el sistema es capaz de replicar datos del nodo maestro al esclavo garantizando la disponibilidad e integridad en los datos, se logra completar de manera efectiva sin errores en el procesamiento y enrutado de los datos. El uso de recursos por parte del sistema para las diferentes configuraciones fue mínimo.No existen conflictos de actualización, conflictos de unicidad, supresión ni de orden, pues al utilizar la replicación

maestro/esclavo de la ONRM al MINBAS no se permiten actualizaciones concurrentes sobre los mismos datos en múltiples sitios.

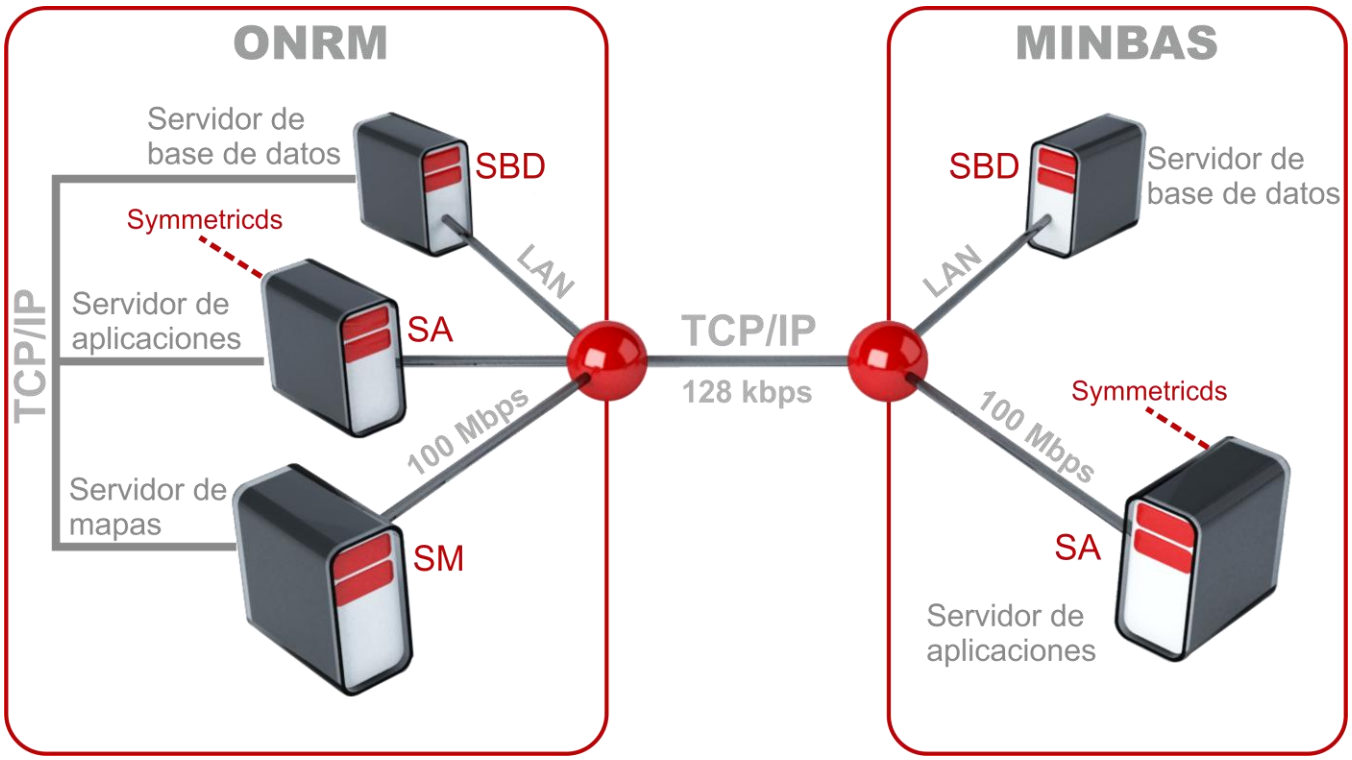


Figura 17: Entorno de Réplica Final.

3.3.3 Configuración de la herramienta.

En principio hay que tener bien claros algunos conceptos que son de vital importancia para la comprensión de cada uno de los parámetros de configuración de SymmetricDS. Se deben identificar los nodos implicados, que no son más que instancias del SymmetricDS. En este caso en particular se cuenta con dos grupos de nodos, el primero con los servidores que se encuentran en la Oficina Nacional de Recursos Minerales y el otro con los servidores que se encuentran en el Ministerio de la Industria Básica en el cual se almacenarán los datos de cada una de las instancias asociadas. Es importante organizar los nodos pues constituyen un factor primordial para la sincronización de los datos, clasificándolos según la información que manejarán y de esta manera definir los grupos de nodos, los cuales son los encargados de encaminar los datos, se definen dos grupos de nodos uno para cada una de las instituciones, incluyendo el nodo central. Aunque los datos que se deben

sincronizar son prácticamente los mismos, cada institución maneja particularidades diferentes, la ruta para garantizar la sincronización de los datos es configurada mediante el grupo de nodos.

Los enlaces entre nodos no son más que la vinculación del nodo origen y destino especificando las acciones a realizar, las mismas pueden ser entrar datos (P) o sacarlos (W). Se han definido dos enlaces de entrada y dos enlaces de salida uno para cada una de las instituciones con la empresa.
Los canales es el medio por el cual viaja la información, en este se definen las prioridades de sincronización de los datos en este caso se ha definido un solo canal, pues cada una de las instituciones asociadas en un momento determinado, puede tener mayor prioridad de sincronización que otra o simplemente la misma, los canales pueden ser habilitados o deshabilitados según los intereses.

Los triggers o disparadores se utilizan para registrar y capturar los cambios. Cada trigger se define teniendo en cuenta una tabla y un canal de transmisión. Además puede especificar los cambios definidos como: actualizar, eliminar o simples inserciones.
Se deben definir cada una de estas características las cuales quedan plasmadas en el archivo de configuración nombrado insert_sample.sql que más adelante se describe como quedaría cada uno de estos parámetros.

Después de explicados algunos de los conceptos fundamentales de SymmetricDS, en esta sección se hace un análisis más profundo de cada uno de los parámetros para su configuración. La herramienta cuenta con varias tablas de configuración en las que se definen los detalles, las más importantes serán explicadas de manera que se pueda realizar una correcta configuración para su entorno de réplica.
Las tablas del SymmetricDS serán creadas en sus inicios vacías pero se configura el archivo insert_sample.sql que es el que realmente guarda las configuraciones del SymmetricDS, este archivo es cargado en el proceso de instalación.

Nodo

Cada nodo representa una localización o una instancia de SymmetricDS que se define en la tabla sym_node en la cual se registran uno a uno de los nodos. Para el nodo central o principal se debe insertar su configuración, mediante las siguientes instrucciones SQL:

```
insert into sym_node (node_id, node_group_id, external_id, sync_enabled)
values ('00000', 'onrm', '00000', 1);
insert into sym_node_identity values ('00000');
```

node_id: Identificador único del nodo, es de tipo varchar.

node_group_id: Identificador del grupo al que pertenece, es de tipo varchar.

external_id: Un identificador de dominio específico para el contexto en el sistema local, es de tipo varchar.

sync_enabled: Es el parámetro que precisa si el nodo estará habilitado (0) o deshabilitado (1) para la sincronización, es de tipo entero.

El *external_id* y *node_group_id* se definen en los archivos ***.properties***.

Grupo de nodos

Los grupos de nodos son sencillos de configurar y se definen en la tabla sym_node_group. Para la propuesta se definieron dos grupos de nodos uno para la Oficina Nacional de Recursos Minerales y otro para el Ministerio de la Industria Básica. Las siguientes instrucciones SQL crearían los grupos de nodos para "ONRM" y "MINBAS".

```
insert into sym_node_group (node_group_id, description)
values ('onrm', 'Servidores que se encuentren dentro de la Oficina Nacional de Recursos
Minerales.');

insert into sym_node_group (node_group_id, description)
values ('minbas', 'Servidores que se encuentren dentro del Ministerio de la Industria Básica');
```

node_group_id: Identificador único para un grupo de nodos.

description: Descripción de este grupo de nodos.

Ambos parámetros se comportan como varchar.

Enlaces

Mediante el enlace, un grupo de nodos envían sus actualizaciones de datos a un grupo de nodos destino. Las siguientes sentencias SQL muestran las acciones de los enlaces entre la ONRM y el MINBAS.

```
insert into sym_node_group_link (source_node_group_id, target_node_group_id,
data_event_action)
values ('onrm', 'minbas', 'P');
```

source_node_group_id: Grupo de nodos en los cuales los cambios deberían ser capturados.

target_node_group_id: Grupo de nodos en cual los cambios de los datos serán enviados.

data_event_action: Esquema de notificación usado para enviar los cambios en los datos al grupo de nodos destinos (en este caso en especifico que se va a replicar la información unidireccionalmente se pone 'P' para entrar los datos solamente del origen al destino).

Canales de Transmisión

Los canales de transmisión se definen en la tabla sym_channel en la cual se puede controlar el máximo de información que será procesada. Permiten además la sincronización de SymmetricDS ya que pueden encontrarse habilitado o deshabilitado. Las siguientes sentencias SQL muestran el canal definido para cada una de las instituciones.

```
insert into sym_channel(channel_id, processing_order, max_batch_size, enabled, description)
values('onrm1', 1, 100000, 1, 'Canal por donde se transmiten los datos desde la oficina ONRM
al MINBAS');
```

channel_id: Identificador único del canal.
processing_order: Prioridad para procesar los datos del canal.

max_batch_size: Número máximo de datos a procesar por este canal.

enabled: Indica si el canal está habilitado o no para la transmisión.

description: Descripción del canal.

Disparadores o Triggers

Los disparadores de SymmetricDS se definen en la tabla sym_trigger. Las siguientes instrucciones SQL definen un disparador que captura los datos de una tabla determinada, cuando estos son insertados, actualizados o eliminados; al no definirse los mismos se asimila que son inicializados en 1 por defecto. A los disparadores se les asigna además el canal por el que se transmitirán los datos.

```
insert into sym_trigger(source_table_name, target_table_name, source_node_group_id,
target_node_group_id, channel_id, sync_on_insert, sync_on_update, sync_on_delete,
excluded_column_names,initial_load_order, last_updated_by, last_updated_time, create_time)
values('access','access','onrm','minbas', 'onrm1', 1, 1, 1, 'aid',100, 'portal', current_timestamp,
current_timestamp);
```

source_table_name: Nombre de la tabla de origen que tendrá un disparador para observar los cambios de los datos.

source_node_group_id: Identificador del grupo de nodos en donde se instalaran los disparadores para registrar los cambios y luego replicarlos.

target_node_group_id: Identificador del grupo de nodos, a los que serán enviados los datos.

sync_on_insert: Se dispara al insertarse algún dato de la tabla, es de tipo entero con valores de 0 o 1.

sync_on_update: Se dispara al actualizarse algún dato de la tabla, es de tipo entero con valores de 0 o 1.

sync_on_delete: Se dispara al eliminarse algún dato de la tabla, es de tipo entero con valores de 0 o 1.

initial_load_order: Orden de secuencia de esta mesa cuando un inicial la carga se envía a un nodo.

last_updated_by: Especifica el usuario que actualizó la última vez esta entrada.

channel_id: Identificador del canal por el cual fluyen los cambios de los datos.

last_update_time: Establece la hora en la que un usuario actualizó la última entrada.

create_time: Establece la hora cuando fue creada la entrada

Para configurar el PostgreSql

SymmetricDS es compatible con nueve gestores de base de datos, para dar respuesta a la solución se analizará con el gestor PostgreSql. Para que la herramienta de réplica funcione correctamente en PostgreSql se debe crear el lenguaje plpgsql y se necesita modificar en el archivo de *postgresql.conf* la línea:

```
custom_variable_classes = 'symmetric'
```

```
CREATE FUNCTION plpgsql_call_handler() RETURNS language_handler AS
        '$libdir/plpgsql' LANGUAGE C;
CREATE FUNCTION plpgsql_validator(oid) RETURNS void AS
        '$libdir/plpgsql' LANGUAGE C;
CREATE TRUSTED PROCEDURAL LANGUAGE plpgsql
        HANDLER plpgsql_call_handler
        VALIDATOR plpgsql_validator;
```

Conclusiones Parciales

- La ONRM es una institución que necesita replicar sus datos.
- Se describe la infraestructura interna de dicha institución así como las características que debe cumplir la propuesta de réplica.
- Se evidencia el despliegue de la solución así como la configuración de la herramienta en el entorno real de la ONRM.
- Se concluye este capítulo reflejando que la propuesta de solución para replicar datos desplegado en la ONRM cumple con los objetivos de la presente investigación, pues brinda información objetiva que transmite visibilidad en torno al proceso de réplica.

Conclusiones

Para el desarrollo del presente trabajo de diploma se realizó un estudio de los procesos y políticas establecidas para el flujo informativo de la ONRM. Se realizó una extensa búsqueda bibliográfica sobre el proceso de réplica en bases de datos distribuidas.

A través del estudio del estado del arte sobre las herramientas de réplica, se obtuvo como resultado en el presente trabajo de diploma una solución de replicación robusta para PostgreSql que garantice la integridad y la disponibilidad de los datos en la Oficina Nacional de Recursos Minerales.

Se realizó una exhaustiva búsqueda bibliográfica acerca de las últimas tendencias y tecnologías que a nivel internacional se están utilizando en el mundo de la informática, para definir aquellas que mejor respuesta darían al problema al cual se pretende dar solución, partiendo de la previa definición del gestor de base de datos PostgreSql y haciendo uso de tecnologías libres.

Este trabajo realiza una contribución al proceso de réplica de bases de datos distribuidas permitiendo realizar de manera más sencilla el proceso de propagar datos de una base de datos a otra con el objetivo de obtener mejor rendimiento y confiabilidad. Las pruebas aplicadas a las diferentes configuraciones demuestran la incapacidad de procesamiento de un servidor de BD centralizadas frente a un sistema distribuido para satisfacer altas concurrencias.

Por tanto se considera que se han cumplido cada uno de los objetivos trazados del presente trabajo, mediante la propuesta de una solución de replicación que permita la configuración, gestión y mantenimiento de una política de réplica de datos ofreciendo alta disponibilidad y rendimiento para las nuevas soluciones de la Oficina Nacional de Recursos Minerales y el MINBAS.

Recomendaciones

El presente trabajo de diploma propone una solución que permite la configuración, gestión y mantenimiento de una política de réplica de BD ofreciendo alta disponibilidad y rendimiento de los servidores de BD para las nuevas soluciones de la ONRM y el MINBAS, por lo que se recomienda:

- ✓ La implementación de una aplicación con interfaz gráfica de usuario que permita la administración y configuración de los servicios de la herramienta SymmetricDS.
- ✓ Profundizar en la teoría expuesta en el documento para uso general en la universidad, tanto para los proyectos productivos como para el uso con fines docentes e investigativos.
- ✓ Realizar un estudio del cual se derive un procedimiento para el diseño e implementación de sistemas de réplica.

Referencias Bibliográficas

[1]. *Alta disponibilidad y rápida recuperación ante desastres para aplicaciones Windows.* [Digital] s.l. : TotemGuard.

[2]. **Hernández León, Rolando Alfredo y Coello Gonzá, Sayda.** *EL PARADIGMA CUANTITATIVO DE LAS INVESTIGACIÓN CIENTÍFICA.* Ciudad de la Habana : EDUNIV, 2002.

[3].Máster en Software Libre. *Máster en Software Libre.* [En línea] Universidad de Extremadura. [Citado el: 3 de 10 de 2009.] http://www.unex.es/eweb/msl/cursos/gestionbbdd.html.

[4]. **Torres, José Miguel.** *SQL Server Compact 2008 SP1 Referencia Completa.* s.l. : Krasis Press, 2008.

[5]. **Culebro Juárez, Montserrat, Gómez Herrera, Wendy Guadalupe y Torres Sánchez, Susana.** *Software libre vs software propietario. Ventajas y desventajas.* México : s.n., Mayo, 2006.

[6]. **Quiñones, Ernesto.** *Las 10 empresas mas grandes del mundo usan Software Libre.* Lima, Perú : s.n., 2009.

[7]. **Powell, Gavin.** *Beginning Database Design.* s.l. : Wiley Publishing, Inc., 2006. ISBN-13: 978-0-7645-7490-0.

[8]. **Mato, Lic. Rosa María García.** *Diseño de Bases de Datos.* 1999.

[9]. **Rodríguez García, Kenny y Baños Fernández, Eddy Ernesto.** *Sistema de réplica para la Intranet Corporativa y el Sitio en Internet de PDVSA.* Ciudad de La Habana : s.n., 2009.

[10]. **Aguiar, Ing Leidy Torres.** *Paquete de Herramientas para la extracción automática de métricas a partir de elementos de configuración del Software.* Ciudad de La Habana : s.n., 2009.

[11]. **Peter Rob, Carlos Coronel.** *Sistemas de bases de datos: Diseño, implementación y administración.* s.l. : Cengage Learning Editores.

[12]. **Pupo Polanco, Rosell y González Pérez, Yenier.** *Implementación del componente réplica de base de datos para Akademos v2.0.* Ciudad de la Habana, Cuba : s.n., 2009.

[13]. **Monge, Raúl.** *Sistemas Distribuidos de Computación. Base de Datos Distribuidas: Replicación.* Valparaíso : Universidad Técnica Federico Santa María, 27 de Junio del 2005.

[14]. **Vega, Norge Fajardo.** *Sistema de réplica para bases de datos distribuidas en PostgreSQL.* La Habana : s.n., 2007.

[15]. **Hende, G.** *Aplicación para resolución de conflictos en bases de datos que no ofrezcan características de distribución de datos.* Bogotá DC : Fundación Universitaria San Martin. , 2005.

[16]. **Bucardo.org.** *Bucardo.org.* [En línea] [Citado el: 3 de 9 de 2009.] http://bucardo.org/bucardo.html.

[17]. **PostgreSQL.** *PostgreSQL.* [En línea] [Citado el: 3 de 9 de 2009.] http://www.postgresql.org.

[18]. **Perl DBI.** *Perl DBI.* [En línea] [Citado el: 3 de 9 de 2009.] http://dbi.perl.org/about/.

[19]. **Mullane, Greg Sabino.** CPAN. *CPAN.* [En línea] [Citado el: 3 de 9 de 2009.] http://search.cpan.org/dist/DBD-Pg/Pg.pm.

[20]. **DavidPashley.com.** *DavidPashley.com.* [En línea] 2006. [Citado el: 3 de 9 de 2009.] http://www.davidpashley.com/articles/perl-io-objects.html.

[21]. **Moose.** *Moose.* [En línea] [Citado el: 3 de 9 de 2009.] http://www.iinteractive.com/moose/.

[22]. **Wall, Larry, Christiansen, Tom y Orwant, Jon.** 32.40. Sys::Hostname. *Programming Perl.* Sebastopol CA, Estados Unidos. : O'Reilly & Associates., Julio, 2000.

[23]. **Ibañez, Marcos.** *WhyFLOSS Conference 5º edición Corrientes Mayo 2008.* 2008.

[24]. **Slony - I.** *Slony - I.* [En línea] 2007. [Citado el: 4 de 9 de 2009.] http://www.slony.info.

[25]. **PgFoundry.** *PgFoundry.* [En línea] 5 de 6 de 2008. [Citado el: 4 de 9 de 2009.] http://pgfoundry.org/projects/pyreplica/.

[26]. **DBReplicator.** *DBReplicator.* [En línea] [Citado el: 5 de 9 de 2009.] http://dbreplicator.org/.

[27]. **SymmetricDS.** *SymmetricDS.* [En línea] 9 de 4 de 2009. [Citado el: 5 de 9 de 2009.] http://symmetricds.codehaus.org/.

[28]. **Long, Eric y Henson, Chris.** *Manual SymmetricDS.* 2009.

[29]. PgCluster. *PgCluster.* [En línea] 2005. [Citado el: 4 de 9 de 2009.] http://pgcluster.projects.postgresql.org/.

[30]. CyberTec. *CyberTec.* [En línea] [Citado el: 4 de 9 de 2009.] http://www.postgresql.at/english/pr_cybercluster_e.html.

[31]. **Estrada, Francys D.** *CONTROLES INDUSTRIALES EN INTERNET.* Venezuela : s.n., 2006.

[32]. **Gómez., Marichelo.** *Conceptos básicos de Bases de datos.* 2008.

[33]. **henry., Kevin.** *Conectividad de la Base de Datos de Java.* 2001.

[34]. **Java SQL Database Engine.** *Java SQL Database Engine.* [En línea] [Citado el: 23 de 10 de 2009.] http://hsqldb.org/

[35]. **H2 database engine.** *H2 database engine.* [En línea] [Citado el: 23 de 10 de 2009.] http://www.h2database.com/html/main.html.

[36]. **Apache Derby.** *Apache Derby.* [En línea] [Citado el: 23 de 10 de 2009.] http://db.apache.org/derby/.

[37]. **Firebird.** *Firebird.* [En línea] [Citado el: 23 de 10 de 2009.] http://www.firebirdsql.org/.

[38]. **Spring Framework.** *Spring Framework.* [En línea] [Citado el: 23 de 10 de 2009.] http://springframework.org/.

[39]. **marioalberto.com.** *marioalberto.com.* [En línea] [Citado el: 24 de 10 de 2009.] http://www.marioalberto.com.mx/articulos/jetty.php.

Bibliografía

- **Martinto, Pedro Carlos Pérez.** *El diseño metodológico de la investigación científica. Teoría de Muestreo: población y muestra. Diseño experimental y métodos.* UCI : s.n.
- **Martinto, Pedro Carlos Pérez.** *El proceso de investigación científica. Estructura. Referencias bibliográficas y Normas.* UCI : s.n.
- **Rolando Alfredo Hernández León, Sayda Coello González. 2002.** *EL PARADIGMA CUANTITATIVO DE LAS INVESTIGACIÓN CIENTÍFICA.* Ciudad de la Habana : EDUNIV, 2002.
- **Mato, Lic. Rosa María García.** Diseño de Bases de Datos. 1999.
- **Long, Eric y Henson, Chris.** Manual SymmetricDS. 2009.
- *Alta disponibilidad y rápida recuperación ante desastres para aplicaciones Windows.* [Digital] s.l. : TotemGuard.
- **Hernández León, Rolando Alfredo y Coello Gonzá, Sayda.** *EL PARADIGMA CUANTITATIVO DE LAS INVESTIGACIÓN CIENTÍFICA.* Ciudad de la Habana : EDUNIV, 2002.
- Máster en Software Libre. *Máster en Software Libre.* [En línea] Universidad de Extremadura. [Citado el: 3 de 10 de 2009.] http://www.unex.es/eweb/msl/cursos/gestionbbdd.html.
- **Torres, José Miguel.** *SQL Server Compact 2008 SP1 Referencia Completa.* s.l. : Krasis Press, 2008.
- **Culebro Juárez, Montserrat, Gómez Herrera, Wendy Guadalupe y Torres Sánchez, Susana.** *Software libre vs software propietario. Ventajas y desventajas.* México : s.n., Mayo, 2006.
- **Quiñones, Ernesto.** *Las 10 empresas mas grandes del mundo usan Software Libre.* Lima, Perú : s.n., 2009.
- **Powell, Gavin.** *Beginning Database Design.* s.l. : Wiley Publishing, Inc., 2006. ISBN-13: 978-0-7645-7490-0.
- **Mato, Lic. Rosa María García.** *Diseño de Bases de Datos.* 1999.
- **Rodríguez García, Kenny y Baños Fernández, Eddy Ernesto.** *Sistema de réplica para la Intranet Corporativa y el Sitio en Internet de PDVSA.* Ciudad de La Habana : s.n., 2009.
- **Aguiar, Ing Leidy Torres.** *Paquete de Herramientas para la extracción automática de métricas a partir de elementos de configuración del Software.* Ciudad de La Habana : s.n., 2009.
- **Peter Rob, Carlos Coronel.** *Sistemas de bases de datos: Diseño, implementación y administración.* s.l. : Cengage Learning Editores.

- **Pupo Polanco, Rosell y González Pérez, Yenier.** *Implementación del componente réplica de base de datos para Akademos v2.0.* Ciudad de la Habana, Cuba : s.n., 2009.
- **Monge, Raúl.** *Sistemas Distribuidos de Computación. Base de Datos Distribuidas: Replicación.* Valparaíso : Universidad Técnica Federico Santa María, 27 de Junio del 2005.
- **Vega, Norge Fajardo.** *Sistema de réplica para bases de datos distribuidas en PostgreSQL.* La Habana : s.n., 2007.
- **Hende, G.** *Aplicación para resolución de conflictos en bases de datos que no ofrezcan características de distribución de datos.* Bogotá DC : Fundación Universitaria San Martin. , 2005.
- **Bucardo.org.** *Bucardo.org.* [En línea] [Citado el: 3 de 9 de 2009.] http://bucardo.org/bucardo.html.
- **PostgreSQL.** *PostgreSQL.* [En línea] [Citado el: 3 de 9 de 2009.] http://www.postgresql.org.
- **Perl DBI.** *Perl DBI.* [En línea] [Citado el: 3 de 9 de 2009.] http://dbi.perl.org/about/.
- **Mullane, Greg Sabino.** CPAN. *CPAN.* [En línea] [Citado el: 3 de 9 de 2009.] http://search.cpan.org/dist/DBD-Pg/Pg.pm.
- **DavidPashley.com.** *DavidPashley.com.* [En línea] 2006. [Citado el: 3 de 9 de 2009.] http://www.davidpashley.com/articles/perl-io-objects.html.
- **Moose.** *Moose.* [En línea] [Citado el: 3 de 9 de 2009.] http://www.iinteractive.com/moose/.
- **Wall, Larry, Christiansen, Tom y Orwant, Jon.** 32.40. Sys::Hostname. *Programming Perl.* Sebastopol CA, Estados Unidos. : O'Reilly & Associates., Julio, 2000.
- **Ibañez, Marcos.** *WhyFLOSS Conference 5º edición Corrientes Mayo 2008.* 2008.
- **Slony - I.** *Slony - I.* [En línea] 2007. [Citado el: 4 de 9 de 2009.] http://www.slony.info.
- **PgFoundry.** *PgFoundry.* [En línea] 5 de 6 de 2008. [Citado el: 4 de 9 de 2009.] http://pgfoundry.org/projects/pyreplica/.
- **DBReplicator.** *DBReplicator.* [En línea] [Citado el: 5 de 9 de 2009.] http://dbreplicator.org/.
- **SymmetricDS.** *SymmetricDS.* [En línea] 9 de 4 de 2009. [Citado el: 5 de 9 de 2009.] http://symmetricds.codehaus.org/.
- **Long, Eric y Henson, Chris.** *Manual SymmetricDS.* 2009.
- PgCluster. *PgCluster.* [En línea] 2005. [Citado el: 4 de 9 de 2009.] http://pgcluster.projects.postgresql.org/.
- CyberTec. *CyberTec.* [En línea] [Citado el: 4 de 9 de 2009.] http://www.postgresql.at/english/pr_cybercluster_e.html.
- **Estrada, Francys D.** *CONTROLES INDUSTRIALES EN INTERNET.* Venezuela : s.n., 2006.

- **Gómez., Marichelo.** *Conceptos básicos de Bases de datos.* 2008.
- **henry., Kevin.** *Conectividad de la Base de Datos de Java.* 2001.
- **Java SQL Database Engine.** *Java SQL Database Engine.* [En línea] [Citado el: 23 de 10 de 2009.] http://hsqldb.org/
- **H2 database engine.** *H2 database engine.* [En línea] [Citado el: 23 de 10 de 2009.] http://www.h2database.com/html/main.html.
- **Apache Derby.** *Apache Derby.* [En línea] [Citado el: 23 de 10 de 2009.] http://db.apache.org/derby/.
- **Firebird.** *Firebird.* [En línea] [Citado el: 23 de 10 de 2009.] http://www.firebirdsql.org/.
- **Spring Framework.** *Spring Framework.* [En línea] [Citado el: 23 de 10 de 2009.] http://springframework.org/.
- **marioalberto.com.** *marioalberto.com.* [En línea] [Citado el: 24 de 10 de 2009.] http://www.marioalberto.com.mx/articulos/jetty.php.

Anexos

Anexo 1 Configuraciones

```
-------------------------------------------------------------------------
-- Sample Symmetric Configuration
-------------------------------------------------------------------------
--
-- Nodes
--
insert into sym_node_group (node_group_id, description)
values ('onrm', 'Servidores que se encuentren dentro de la Oficina Nacional de
Recursos Minerales.');
insert into sym_node_group (node_group_id, description)
values ('minbas', 'Servidores que se encuentren dentro del Ministerio de la Industria
Básica');

insert into sym_node_group_link (source_node_group_id, target_node_group_id,
data_event_action)
values ('onrm', 'minbas', 'P');

insert into sym_node (node_id, node_group_id, external_id, sync_enabled)
values ('00000', 'onrm', '00000', 1);
insert into sym_node_identity values ('00000');

--
-- Channels
--
insert into sym_channel(channel_id, processing_order, max_batch_size, enabled,
description)
values('onrm1', 1, 100000, 1, 'Canal por donde se transmiten los datos desde la oficina
ONRM al MINBAS');

--
-- Triggers
--
insert into sym_trigger(source_table_name, target_table_name, source_node_group_id,
target_node_group_id, channel_id, sync_on_insert, sync_on_update, sync_on_delete,
excluded_column_names,initial_load_order, last_updated_by, last_updated_time,
create_time)
values('access','access','onrm','minbas', 'onrm1', 1, 1, 1, 'aid',100, 'portal',
current_timestamp, current_timestamp);
```

Configuración 2

```
-------------------------------------------------------------------------
-- Sample Symmetric Configuration
-------------------------------------------------------------------------
--
-- Nodes
--
insert into sym_node_group (node_group_id, description)
values ('onrm', 'Servidores que se encuentren dentro de la Oficina Nacional de
Recursos Minerales.');
insert into sym_node_group (node_group_id, description)
values ('minbas', 'Servidores que se encuentren dentro del Ministerio de la Industria
Básica');

insert into sym_node_group_link (source_node_group_id, target_node_group_id,
data_event_action)
values ('onrm', 'minbas', 'P');

insert into sym_node (node_id, node_group_id, external_id, sync_enabled)
values ('00000', 'onrm', '00000', 1);
insert into sym_node_identity values ('00000');

--
-- Channels
--
insert into sym_channel(channel_id, processing_order, max_batch_size, enabled,
description)
values('onrm1', 1, 100000, 1, 'Canal por donde se transmiten los datos desde la oficina
ONRM al MINBAS');

--
-- Triggers
--

insert into sym_trigger(source_table_name, target_table_name, source_node_group_id,
target_node_group_id, channel_id, sync_on_insert, sync_on_update, sync_on_delete,
excluded_column_names,initial_load_order, last_updated_by, last_updated_time,
create_time)
values('balancea.tcontrolexterno','balancea.tcontrolexterno','onrm','minbas', 'onrm1', 1,
1, 1, 'idcontrolext',100, 'portal', current_timestamp, current_timestamp);
```

Configuración 3

```
-------------------------------------------------------------------------
-- Sample Symmetric Configuration
-------------------------------------------------------------------------
--
-- Nodes
--
insert into sym_node_group (node_group_id, description)
values ('onrm', 'Servidores que se encuentren dentro de la Oficina Nacional de
Recursos Minerales.');
insert into sym_node_group (node_group_id, description)
values ('minbas', 'Servidores que se encuentren dentro del Ministerio de la Industria
Básica');

insert into sym_node_group_link (source_node_group_id, target_node_group_id,
data_event_action)
values ('onrm', 'minbas', 'P');

insert into sym_node (node_id, node_group_id, external_id, sync_enabled)
values ('00000', 'onrm', '00000', 1);
insert into sym_node_identity values ('00000');

--
-- Channels
--
insert into sym_channel(channel_id, processing_order, max_batch_size, enabled,
description)
values('onrm1', 1, 100000, 1, 'Canal por donde se transmiten los datos desde la oficina
ONRM al MINBAS');

--
-- Triggers
--

insert into sym_trigger(source_table_name, target_table_name, source_node_group_id,
target_node_group_id, channel_id, sync_on_insert, sync_on_update,
sync_on_delete,initial_load_order, last_updated_by, last_updated_time, create_time)
values('balancem.tcalidadum_rec','balancem.tcalidadum_rec','onrm','minbas', 'onrm1',
1, 1, 1,100, 'portal', current_timestamp, current_timestamp);
```

Configuración 4

```
-------------------------------------------------------------------------
-- Sample Symmetric Configuration
-------------------------------------------------------------------------
```

```
--
-- Nodes
--
insert into sym_node_group (node_group_id, description)
values ('onrm', 'Servidores que se encuentren dentro de la Oficina Nacional de
Recursos Minerales.');
insert into sym_node_group (node_group_id, description)
values ('minbas', 'Servidores que se encuentren dentro del Ministerio de la Industria
Básica');

insert into sym_node_group_link (source_node_group_id, target_node_group_id,
data_event_action)
values ('onrm', 'minbas', 'P');

insert into sym_node (node_id, node_group_id, external_id, sync_enabled)
values ('00000', 'onrm', '00000', 1);
insert into sym_node_identity values ('00000');

--
-- Channels
--
insert into sym_channel(channel_id, processing_order, max_batch_size, enabled,
description)
values('onrm1', 1, 100000, 1, 'Canal por donde se transmiten los datos desde la oficina
ONRM al MINBAS');

--
-- Triggers
--

insert into sym_trigger(source_table_name, target_table_name, source_node_group_id,
target_node_group_id, channel_id, sync_on_insert, sync_on_update, sync_on_delete,
excluded_column_names,initial_load_order, last_updated_by, last_updated_time,
create_time)
values('balancep.tacumulados','balancep.tacumulados','onrm','minbas', 'onrm1', 1, 1, 1,
'idacumulados',100, 'portal', current_timestamp, current_timestamp);
```