UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO DE UN SISTEMA DE REPLICACIÓN Y DISTRIBUCIÓN DE CONSULTAS EN BASES DE DATOS INFOBRIGHT

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

RONALD ELÍAS POILLOT CARTES

PROFESOR GUÍA:
SR. EDGARD PINEDA LEONE

MIEMBROS DE LA COMISIÓN:
SR. JORGE PÉREZ ROJAS
SR. DIONISIO GONZÁLEZ GONZÁLEZ

SANTIAGO DE CHILE
ABRIL 2012

Resumen

La Inteligencia de Negocios (BI) y la Minería de Datos (DM) son áreas que han tenido un importante crecimiento en los últimos diez años. Estas disciplinas ayudan en la toma de decisiones de las compañías, entregando valiosa información extraída desde los datos de la empresa mediante complejos procedimientos. Resulta indispensable para los proveedores de servicios en estas áreas el contar con los datos disponibles la mayor cantidad de tiempo posible, para no incurrir en quiebres importantes de su continuidad operacional.

El objetivo de la presente memoria fue crear un sistema básico de alta disponibilidad que incluye replicación y distribución de consultas a Bases de Datos, el cual pretende disminuir los riesgos asociados al *downtime* de los servidores de datos de la empresa Penta Analytics. Esta compañía utiliza Infobright como motor de almacenamiento de datos, el cual está orientado a Bases de Datos de tipo analítica y que actualmente no cuenta con una solución de alta disponibilidad que abarque ambos temas.

La solución fue diseñada en base a tres componentes principales: un sistema de replicación de datos, un sistema de distribución de consultas y una tabla de estados de replicación. Para la replicación se desarrolló una aplicación ad-hoc programada en lenguaje Java, mientras que la distribución de consultas fue creada en base a una aplicación llamada *MySQL-Proxy*, que fue adaptada para funcionar con un *clúster* de Bases de Datos analíticas. La tabla de replicación resguarda el estado de actualización de cada componente del sistema, estado que debe ser consistente ante cualquier escenario.

Como resultado se obtuvo un sistema que mejora los tiempos de respaldo de datos y de respuestas a consultas a Bases de Datos. La solución para la distribución de consultas es escalable y paralelizable, mientras que el sistema de replicación escala sólo verticalmente y requiere modificación de código para agregar paralelismo y escalabilidad horizontal.

Este desarrollo constituye una innovación como solución de alta disponibilidad para este tipo de Bases de Datos. En base a la presente memoria se muestra un conjunto de posibles mejoras y trabajos futuros, como por ejemplo mejorar los algoritmos de selección de servidor en la distribución de consultas u optimizar el transporte de datos entre servidores del sistema de replicación.

Agradecimientos

A mi familia, Gastón, Fresia, Yeny y Maxi, quienes han sido y serán un pilar fundamental en mi vida. Les agradezco de corazón todo lo que me han entregado, por brindarme las herramientas necesarias para poder crecer y desarrollarme como persona y como profesional. El ejemplo que me han dado de perseverancia, sacrificio y por sobre todo amor, es invaluable y lo atesoro como el regalo más preciado que he recibido. Papá, ¡misión cumplida!... ya salimos de esto.

A Renata, mi amada compañera, quien ha estado a mi lado pacientemente en momentos difíciles de mi vida y me entregó su incondicional apoyo y comprensión durante el transcurso de este trabajo. Gracias por todo lo que me has entregado, mi corazón se rinde a tus pies.

No puedo dejar de nombrar a mis queridos tíos Héctor y Erly, quienes durante años me hicieron sentir como uno más de sus hijos; tío, sé que estarás orgulloso de este logro allá en el cielo.

A Edgard, mi profesor guía, a quien le debo el crédito de idear este tema de memoria y me guió durante este trabajo con sabios consejos. Gracias por la oportunidad, por las correcciones, por el tiempo dedicado, por los almuerzos con una amena conversación y por compartir tus experiencias conmigo.

A mis amigos de la vida, mis compañeros del CADCC 2009, mis ex compañeros del Instituto Nacional, mis futboleros compinches de *4DIN*, *Unión San Beauchef*, *Reboot*, *Escuela de Injeniería* y tantos otros equipos más. Gracias por el cariño que me entregan, gracias por su amistad.

Al área de tecnologías de Penta Analytics, grandes compañeros de trabajo con quienes compartir el día a día es una experiencia gratificante.

Finalmente quiero registrar una frase que me enseñaron en el año 1997, "*Labor Omnia Vincit*". Este trabajo es un ejemplo concreto de su veracidad.

Ronald Poillot Cartes
Abril del 2012

Índice General

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

1.1. Antecedentes generales

Un componente crítico para el éxito en las empresas modernas es la habilidad de tomar ventaja de toda la información que tenga disponible, tanto externa como interna; el contar con el conocimiento oportuno y preciso puede ser traducido en una mejora en el desempeño de la compañía, el cual a la vez redunda en una mejor rentabilidad para ésta. Por otro lado, este desafío puede llegar a ser bastante dificultoso dado el creciente volumen de los datos a recopilar y analizar [CKKS02].

En este contexto, dos tecnologías han sido pilares para mejorar el valor cualitativo y cuantitativo del conocimiento disponible para las compañías: la Inteligencia de Negocios y la Minería de Datos.

Se conoce como Inteligencia de Negocios, Inteligencia Empresarial o BI (del inglés *Business Intelligence*) al conjunto de estrategias y herramientas enfocadas a la administración y creación de conocimiento mediante el análisis de datos existentes en una empresa; por otro lado, la Minería de Datos (del inglés *Data Mining*) prepara, sondea y explora los datos para sacar la información oculta e implícita en ellos. Bajo este último concepto se engloba todo un conjunto de técnicas encaminadas a la extracción de conocimiento procesable, implícito en las Bases de Datos de las compañías.

Chile no se ha quedado atrás con esta tendencia mundial. Según el Reporte Anual de Business Intelligence[1], en el año 2008 dos de cada tres empresas ya habían implementado

[1] `http://www.cetiuc.cl/wp-content/uploads/2009/03/reporte-anual-de-business-intelligence.pdf`

alguna iniciativa de BI, no existiendo diferencias por rubro en el nivel de adopción. Lo anterior refleja el interés por incorporar estas tecnologías a la industria.

Desde el año 2002 Penta Analytics está presente en el mercado nacional e internacional de la Inteligencia de Negocios y Minería de Datos. Sus servicios buscan mejorar los resultados económicos y la rentabilidad de las empresas, mediante el diseño e implementación de acciones comerciales basadas en la información extraída de sus clientes [Pent11].

El procesamiento de datos es componente fundamental del núcleo del negocio de Penta Analytics y resulta de vital importancia contar con una adecuada estrategia de almacenamiento y resguardo de la información.

La estrategia utilizada actualmente consta de un conjunto de servidores de Bases de Datos, para almacenar la información de los clientes, y un mecanismo de copia de seguridad semanales de estos servidores, para mantener un respaldo de los datos. Cabe destacar que cada servidor de Base de Datos guarda información de múltiples clientes, es decir, no existe un servidor dedicado para cada uno de ellos.

Esta estrategia deja en evidencia un punto singular de falla en los servidores de Bases de Datos. Un servidor caído puede tomar horas en restablecer su completa operatividad, para lo cual debe pasar por un proceso de restauración de datos desde los respaldos semanales. En caso de una pérdida completa del servidor (falla de hardware, por ejemplo) el proceso anterior restaurará los datos de la última copia semanal y no existe mecanismo para recuperar los datos modificados con fecha más reciente a la del respaldo.

El costo para Penta Analytics asociado a la caída de un servidor de Base de Datos es cuantioso. La cadena productiva de la empresa comienza en el procesamiento de los datos de los clientes y si un servidor de Base de Datos está fuera de servicio, esta cadena se ve detenida. Además, como cada servidor contiene datos de más de un cliente, el tener un servidor no operacional se traduce en muchos procesos productivos detenidos, lo cual implica un alto costo económico para la empresa.

Dada la criticidad del manejo y resguardo de la información para el negocio, nace la necesidad de contar con un sistema de alta disponibilidad[2] para el almacenamiento de datos.

[2]Se entiende por alta disponibilidad al nivel de disponibilidad implícito en el diseño de un sistema, que entregue un determinado nivel de continuidad operacional [MaSt03].

Un correcto diseño de un sistema de estas características asegurará, con un alto grado de confiabilidad, que los datos estarán disponibles al momento de iniciar los procesos productivos de la empresa, aún en el caso de ocurrir eventualidades en algún equipo.

Una estrategia comúnmente utilizada para incrementar la disponibilidad de un sistema es la redundancia[3]. Para sistemas de Bases de Datos, esto se traduce en duplicar el contenido de cada servidor en uno o más servidores secundarios. La redundancia puede ser obtenida mediante un sistema que ejecute un proceso de replicación, el cual realiza el trabajo de compartir la información entre distintos servidores y asegurar la consistencia de los datos compartidos entre éstos.

Para aprovechar la ventaja de contar con la información duplicada en más de un servidor es que se utilizan sistemas que brinden distribución de consultas[4]. En este caso, el sistema que distribuya las consultas se debe comunicar con el proceso de replicación, para conocer cuáles servidores son los adecuados para responder una determinada consulta.

El presente tema de memoria presenta el diseño e implementación de una sistema básico de alta disponibilidad para los servidores de Bases de Datos de Penta Analytics, que cubre las necesidades de replicación de Bases de Datos y distribución de consultas.

1.2. Motivación

El no contar con una adecuada estrategia de alta disponibilidad se traduce en una mayor probabilidad de pérdida de información y en un alto costo económico para Penta Analytics, pues en este escenario la empresa no dispone de los datos para brindar sus servicios [MaSt03]. Estos costos pueden catalogados como directos o indirectos; por ejemplo, la pérdida de productividad de los empleados de la empresa y las posibles horas extras utilizadas para cumplir con los plazos de entrega de los reportes corresponden a costos directos; por otro lado, la pérdida en la satisfacción del cliente y la mala publicidad por no prestación de servicios son costos indirectos.

Este tema de memoria busca aplacar los problemas, costos y riesgos asociados al *down-*

[3]Se conoce por redundancia a la duplicación de componentes críticos, con el objetivo de aumentar la fiabilidad de un sistema.

[4]La distribución de consultas se refiere al problema de seleccionar, desde un conjunto de recursos disponibles, una fuente que responda ante una consulta dada [XYEW98].

time[5] de las Bases de Datos. Su objetivo es aumentar el período en que éstas se encuentran operativas, a través del diseño e implementación de un sistema de alta disponibilidad que incluya replicación y distribución de consultas sobre Bases de Datos.

Un servicio de replicación de datos mejora el desempeño total de un sistema y aumenta su disponibilidad, ya que provee fuentes alternativas de acceso a los datos [MPGP01]. Además permite la recuperación de una Base de Datos fuera de servicio en menor tiempo, en comparación con la situación actual. Esto supone una disminución notable de interrupciones de la continuidad de las funciones operacionales de Penta Analytics.

Por su lado, la distribución de consultas permite equilibrar la carga[6] entre distintos servidores, lo cual implica un mejor aprovechamiento de los recursos de hardware disponibles en la empresa y en una mejora en los tiempos de respuesta globales en las consultas a Bases de Datos [RBSc00].

La replicación y la distribución de consultas son temas que han sido abordados con anterioridad en la industria y existen diversas soluciones (aplicaciones) ya desarrolladas (comerciales, de código libre, gratuitas, etc.), tales como Tungsten, MySQL-Proxy, Shard-Query, entre otras, pero cuyo alcance está acotado a ciertos motores específicos de Bases de Datos. En Penta Analytics se utiliza Infobright[7] como motor de Base de Datos, el cual en la actualidad no cuenta con una solución de alta disponibilidad que incluya ambos temas.

Infobright es un motor de Bases de Datos de tipo *analítica*. Este tipo de Bases de Datos son comúnmente utilizadas en las empresas de Inteligencia de Negocios y Minería de Datos y se caracterizan por estar diseñadas para atender en su mayoría a operaciones de lectura de datos. Las operaciones de actualización tienen lugar en períodos bien definidos, siendo éstas una pequeña fracción de tiempo en comparación a las operaciones de lectura. El contraste a este tipo de Bases de Datos son las *transaccionales*, las cuales son diseñadas para trabajar en mayor número con operaciones de actualización por sobre las de lectura.

Para este trabajo de memoria se presenta entonces el desafío de diseñar e implementar sistemas de replicación de datos y de distribución de consultas, los cuales han de interactuar

[5]Se conoce por *downtime* al período de tiempo en que un sistema computacional, o parte de éste, no se encuentra operativo.

[6]La carga de un sistema es una medida de la cantidad de trabajo que el sistema realiza.

[7]`http://www.infobright.com/`.

con un clúster[8] de Bases de Datos analíticas. La solución provista en este trabajo representa una innovación en la industria y el primer ejemplo de integración de alta disponibilidad en un esquema de almacenamiento de datos con Infobright.

1.3. Objetivo general

El objetivo general de este tema de memoria es diseñar e implementar un sistema de alta disponibilidad que brinde replicación y distribución de consultas, el cual opere sobre las Bases de Datos y servidores de Penta Analytics.

El esquema sobre el cual operará este sistema será un clúster o grupo de servidores de Bases de Datos analíticas, con un servidor *maestro* de datos y uno o más servidores *esclavos*. El motor de Base de Datos con el cual debe interactuar el sistema es Infobright.

Los objetivos específicos de este trabajo son los siguientes:

- Diseñar e implementar un sistema que brinde replicación sobre Bases de Datos a nivel de tablas.
- Diseñar e implementar un sistema que distribuya consultas sobre distintos servidores de Bases de Datos.
- Medir el desempeño del sistema ante diferentes escenarios.

1.4. Metodología de Trabajo

La metodología definida para el desarrollo de este trabajo de memoria fue la siguiente:

- Estudio de técnicas de replicación y distribución de consultas sobre Bases de Datos; estudio de ventajas y desventajas en cuanto a desempeño total del sistema y tiempos de respuesta.
- Levantamiento de situación actual en Penta Analytics: estudio de Bases de Datos, estructura de redes y servidores.

[8]Un clúster consiste en un conjunto de computadores interconectados que al trabajar de forma tan cercana y relacionada, en muchos aspectos pueden ser considerados como sólo un sistema.

- Definición del estado del arte de aplicaciones ya desarrolladas que ofrezcan soluciones de replicación y distribución de consultas, específicamente para el motor de Base de Datos MySQL. Esto se debe a la alta similitud entre el motor mencionado y la estructura de Infobright [Info10]. Es importante considerar en el análisis la aplicabilidad de la solución propuesta a un clúster de Bases de Datos de tipo analítica.

- Diseño de una estrategia adecuada de replicación y distribución de consultas en un clúster de Bases de Datos Infobright.

- Definición de requisitos del sistema, escenarios de ejecución y pruebas de validación del trabajo realizado. Se deben especificar los tipos de interacción, el protocolo y los casos de uso para las aplicaciones de replicación y distribución de consultas.

- Implementación de sistema de replicación de las Bases de Datos.

- Implementación de sistema de distribución de consultas a las Bases de Datos.

- Realización de pruebas de la solución y medición de desempeño contrastando los tiempos obtenidos con el sistema en relación al escenario sin la presencia de éste.

- Definición de posibles mejoras en base a las pruebas y mediciones obtenidas.

1.5. Organización de la memoria

El informe está organizado en cinco capítulos, de los cuales el primero de éstos corresponde al actual.

En el capítulo 2 se presentan diversos antecedentes relacionados con el trabajo realizado, incluyendo definiciones sobre conceptos utilizados a lo largo de este informe, el resultado de la investigación previa al trabajo y el levantamiento de la situación actual de la empresa.

En el capítulo 3 se describe el diseño de alta disponibilidad creado y la implementación de los sistemas de replicación y distribución de consultas.

En el capítulo 4 se presentan los resultados y las conclusiones sobre el trabajo realizado, para finalizar con el capítulo 5 en donde se describen posibles aplicaciones y trabajos a realizar en el futuro a partir de la solución creada.

Capítulo 2

Antecedentes

En este capítulo se plasma una investigación bibliográfica realizada acerca de los conceptos, marco teórico y aplicaciones asociadas al sistema de alta disponibilidad y el motor de Base de Datos sobre la cual se trabajará; esto con el fin de considerar la estrategia que mejor se adapte a la situación actual de Penta Analytics y para encontrar componentes y características que puedan ser reutilizadas tanto para la replicación como para la distribución de consultas.

2.1. Replicación de Bases de Datos

Se entiende por replicación de Bases de Datos al proceso de creación y mantención de versiones duplicadas de objetos de Bases de Datos en un sistema distribuido [May10]. Como consecuencia de este proceso se mejora el desempeño total del sistema y se incrementa la disponibilidad al proveer rutas alternativas de acceso a los datos [MPGP01].

2.1.1. Tipos de Replicación

Una estrategia de replicación puede ser seleccionada basada en distintas características. Por ejemplo, se puede realizar una categorización discriminando en qué momento se efectúan las actualizaciones en los servidores, en cuáles de éstos serán realizadas, el grado de comunicación entre sí que tendrán las Bases de Datos, la forma en que se declarará que una transacción se ha ejecutado completamente en el sistema, etc.

A continuación se revisarán las distintas clasificaciones de los métodos de replicación según las características antes mencionadas.

Replicación síncrona y asíncrona

Se dice que un tipo de replicación es **eager**, **temprana** o **síncrona** cuando las actualizaciones son propagadas de manera sincronizada desde el servidor principal a los secundarios; en cambio, cuando este evento sucede de manera asíncrona se dice que es un tipo de replicación **lazy**, **tardía** o **asíncrona**.

Un sistema de replicación síncrona se basa en el principio del protocolo de ejecución en dos fases (*two-phase commit protocol*, 2PC) [BeHG87]. Cuando una actualización se ejecuta en la Base de Datos *maestra*, el sistema se conecta con los clientes de ésta, las bloquea a nivel de registro y luego realiza la actualización simultánea en todas las Bases de Datos *esclavas*.

Si alguna Base de Datos no está disponible, la actualización no se ejecuta y la consistencia de los datos se mantiene. Por tanto, para una correcta ejecución de la operación se requiere de la disponibilidad de todos los servidores involucrados al momento de la propagación de la actualización.

Un sistema de replicación asíncrona no propaga de inmediato las actualizaciones realizadas en el sistema. Existe un tiempo de retraso entre la ejecución de la instrucción de actualización y la efectiva propagación de los resultados de ésta a las réplicas del clúster, el cual debe ser ajustado para cada solución. Se espera que los relojes de los servidores se encuentren sincronizados o cercanamente sincronizados. La correctitud de la operación no depende de esto último, pero el desempeño del sistema puede verse afectado cuando la diferencia entre estos relojes es grande [LLSG92].

Existen dos variantes de replicación asíncrona: **periódica** y **sin período**. Con un sistema periódico de replicación, las actualizaciones en los servidores secundarios son ejecutadas a intervalos específicos de tiempo; en tanto con un sistema de replicación sin período las actualizaciones son propagadas sólo cuando son necesarias (generalmente basadas en un evento generado en un trigger[1]). Existe un intervalo de tiempo en donde los datos no son consistentes, el cual debe ser controlado y ajustado según las necesidades de la aplicación.

[1]Un trigger es un procedimiento escrito en algún lenguaje de programación (SQL, en este caso) que se ejecuta cada vez que existe alguna modificación en una tabla, vista o cuando tienen lugar algunas acciones específicas del usuario o del sistema de Base de Datos [Orac11].

Copia Primaria y Actualización en todos lados

Según el lugar donde se realiza la actualización la replicación se puede clasificar como Copia Primaria (Primary Copy) o Actualización en todos lados (Update Anywhere). En el primer tipo existe sólo un lugar donde es posible ejecutar una actualización (llamada *copia primaria* o Base de Datos *maestra*) y los clientes o *esclavos* son actualizados reflejando los cambios realizados en la Base *maestra*; en cambio en la segunda clasificación las actualizaciones pueden ser ejecutadas y propagadas por cualquiera de las Bases de Datos del clúster.

Terminación Votada y No Votada

Dependiendo de cómo sea establecido el protocolo para determinar cómo terminar una transacción se puede realizar una clasificación en dos tipos: terminación Votada y No Votada (*Voting and Non-voting termination*).

En el caso de la terminación Votada se requiere una ronda extra de mensajes para coordinar las diferentes réplicas. Esta ronda puede ser tan compleja como un protocolo de ejecución atómico (atomic commitment protocol, como por ejemplo el protocolo de terminación de dos fases 2PC [BeHG87]), o tan simple como un mensaje de confirmación enviado por los servidores de Bases de Datos.

En cambio, la terminación No Votada implica que cada sitio puede decidir por sí solo si ejecutar o abortar una transacción. Las técnicas que no involucran votación requieren que las réplicas se comporten de manera determinista; en otras palabras que desde cierto punto de la ejecución en adelante la Base de Datos presente el mismo comportamiento y ejecute las mismas instrucciones siempre.

2.1.2. Estrategias de Replicación

A continuación se presenta un listado de los principales tipos de estrategias de replicación y sus respectivas descripciones:

Replicación vía Snapshot

En este tipo de replicación un snapshot (una imagen instantánea o copia de los datos de la Base de Datos) es obtenida desde un servidor y trasladada a otro servidor o a otra Base

de Datos en el mismo servidor. A pesar de ser la estrategia más simple de replicación tiene el costo asociado de trasladar todos los datos cada vez que una Base de Datos es actualizada.

Replicación Transaccional

En este tipo de replicación el agente replicador monitorea al servidor principal en busca de cambios en la Base de Datos y transmite los cambios al resto de los servidores [Paul08]. La transmisión de los datos se puede hacer de manera instantánea o de forma periódica. Usualmente esta estrategia de replicación es utilizada en escenarios donde la interacción es de servidor a servidor.

Replicación vía Merge

Este tipo de replicación permite que cada una de las réplicas trabaje de manera independiente [Paul08]. Éstas pueden trabajar sin tener conexión entre sí y cuando estén conectadas el agente replicador chequea los cambios que se produjeron en ambos conjuntos de datos y modifica cada Base de Datos de manera acorde. En caso de existir conflictos se utiliza un algoritmo de resolución predefinido para alcanzar la consistencia de ambas Bases de Datos. Esta estrategia se utiliza con frecuencia en ambientes con conexiones inalámbricas.

Replicación basada en Statements

En este tipo de replicación se intercepta cada consulta SQL y la envía a las diferentes réplicas. Cada réplica opera de manera independiente. Las consultas de lectura y escritura son enviadas a todos los servidores, en cambio las consultas de sólo lectura son enviadas a sólo un servidor. Se utiliza esta estrategia en ambientes donde se asume que cada réplica mantendrá un caché prácticamente idéntico al resto de las réplicas.

2.1.3. Aplicaciones de Replicación

Existen diversas aplicaciones, tanto comerciales, gratuitas y de código abierto, que realizan replicación de Bases de Datos. A continuación se describe el resultado de una investigación sobre las distintas herramientas de replicación existentes en el mercado que son compatibles con Bases de Datos MySQL.

Tungsten Replicator

Tungsten Replicator es una herramienta de replicación de datos de alto rendimiento, de código libre y diseñada para Bases de Datos MySQL [Conti11]. Este proyecto es auspiciado por la compañía Continuent Inc.[2] y es la base de Tungsten Enterprise, un producto comercial de clusterización de Bases de Datos.

Esta aplicación está programada en lenguaje Java y trabaja como un software complementario al servidor de Base de Datos. El tipo de replicación soportada es vía copia primaria (*maestro-esclavo*), en donde las operaciones de actualización son manejadas por sólo un servidor de Bases de Datos (servidor primario o *maestro*) y cada actualización se propaga automáticamente a las réplicas (servidores *esclavos* o secundarios).

Tungsten trabaja a través de replicación basada en logs[3]. En este tipo de replicación se leen las instrucciones SQL[4] desde el archivo de log de recuperación de la Base de Datos, el cual contiene una lista serializada de cambios que la Base de Datos utiliza para recuperarse en caso de un reinicio. Al contrario de la actualización mediante triggers, el cual instala triggers para capturar los cambios en las tablas, la replicación basada en logs no representa un costo extra en desempeño para la Base de Datos.

Entre las características destacadas se cuentan: Número identificador único (Id) de transacción global, filtros flexibles sobre transacciones, obtención de metadatos extensible acerca de las transacciones, replicación de datos entre diferentes versiones de Bases de Datos, replicación paralela y también permite replicación con más de una Base de Datos *maestra*.

En la figura 2.1 se muestra la arquitectura del sistema de replicación Tungsten.

Gearman Server

Gearman es un framework de aplicaciones Open Source genérico que permite delegar trabajo a otras máquinas o procesos que puedan realizarlo de forma apropiada y de manera más eficiente. Permite la ejecución de procesos en paralelo, balancear carga de procesamiento

[2] `http://www.continuent.com`.

[3] Un archivo de log es una archivo que contiene mensajes sobre un sistema o proceso. En MySQL existen distintos tipos de logs, los cuales son creados en el directorio de datos de la aplicación [RedH11] [Oracl11].

[4] Structured Query Language (SQL) es un lenguaje especializado para actualizar, borrar y consultar información almacenada en Bases de Datos. SQL es un estándar ANSI e ISO, y es de facto el lenguaje estándar de consulta a Bases de Datos [Indi11].

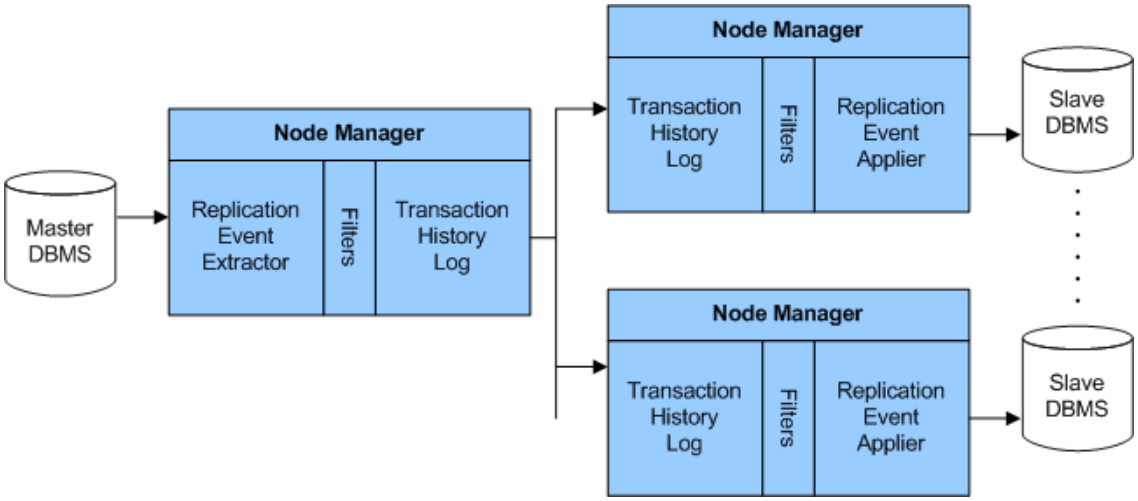

Figura 2.1: Arquitectura del sistema Tungsten. Fuente: [Cont11]

y hacer llamados a funciones entre distintos lenguajes de programación [CoGear11].

El framework Gearman puede ser utilizado por una gran variedad de aplicaciones, desde sitios con alta disponibilidad hasta el transporte de eventos de replicación para Bases de Datos. En otras palabras es el sistema nervioso para la comunicación de sistemas distribuidos.

Entre las características destacadas de Gearman se encuentran:

- Es un framework multilenguaje. Existen interfaces de Gearman implementadas en distintos lenguajes de programación y cada vez aparece nuevas interfaces.
- Es un framework flexible, lo cual permite desarrollar sin basarse ni comprometerse con un patrón de diseño específico.
- Es un framework incrustable, lo cual permite que sea utilizado por aplicaciones de cualquier tamaño. Esta característica también permite que Gearman sea fácil de introducir en aplicaciones ya desarrolladas con un mínimo de costo adicional.
- Gearman tiene protocolos e interfaces simples, con un servidor optimizado programado en lenguaje C, lo cual brinda gran rapidez.

La figura 2.2 muestra el funcionamiento general del framework Gearman.

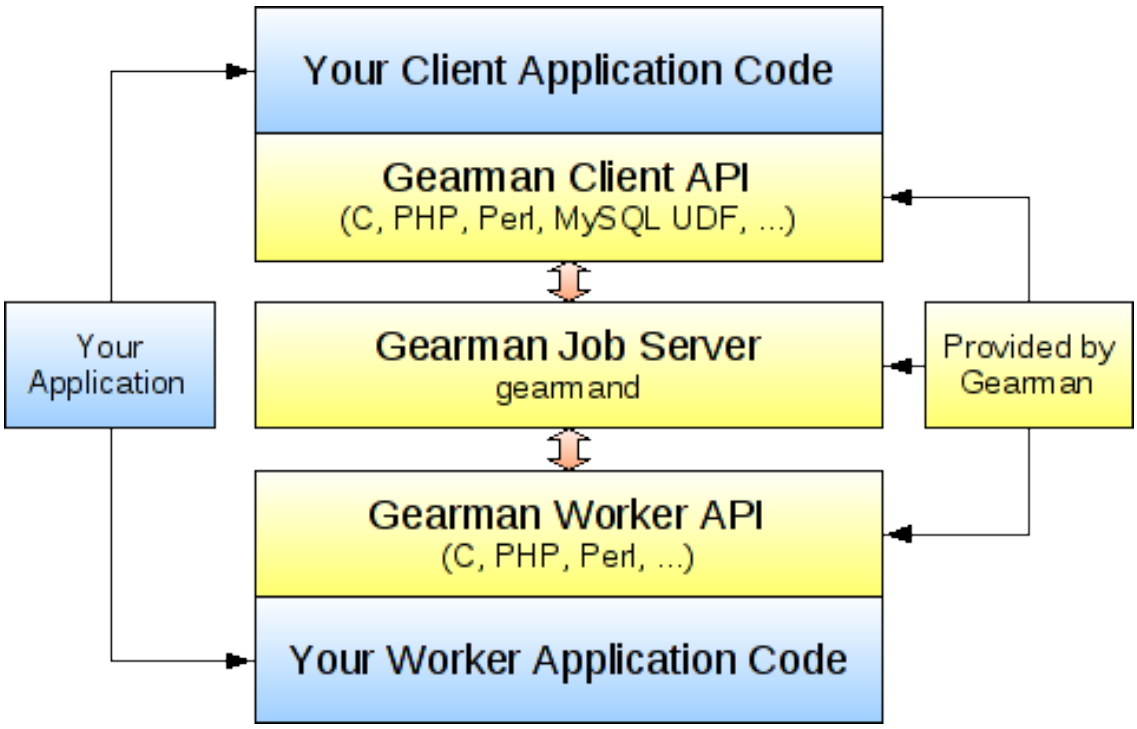

Figura 2.2: Esquema de funcionamiento del framework Gearman. Fuente: [CoGearm11]

2.2. Distribución de consultas

La distribución (encaminamiento, enrutamiento o ruteo) de consultas busca como objetivo mejorar el desempeño general del sistema. Este objetivo se logra al balancear la carga del sistema entre los servidores o computadores que lo componen.

El balanceo de carga es una conocida técnica para utilizar los recursos de un sistema de manera más efectiva mediante la utilización y distribución de tareas particionadas de acuerdo a la estrategia de distribución de carga utilizada [BSDO04]. Este concepto es aplicado en múltiples campos de la computación, tanto en clúster de computadores como en las redes de internet, en unidades de disco, en centrales de procesamiento, etc. Existen también múltiples algoritmos diseñados para lograr un adecuado balanceo de carga.

2.2.1. Algoritmos de Balanceo de Carga

En esta sección se presenta una breve descripción de algunos algoritmos de balanceo de carga existentes actualmente [BSDO04].

Es importante destacar que una estrategia de distribución de carga se dice **adaptativa**

o **no adaptativa** si las condiciones de carga del sistema y sus componentes en el momento de ejecución del algoritmo influyen en las decisiones realizadas por éste. En caso de que sean influyentes se dice que la estrategia es adaptativa; en caso contrario se dice que la estrategia es no adaptativa.

Round-Robin

Este algoritmo de selección de recursos es una estrategia no adaptativa de elección de servidor. En su forma básica el proceso de selección se basa en lo siguiente: Se mantiene un listado de posibles elecciones (servidores, máquinas) y un puntero o contador que hace referencia a uno de esos recursos. Cuando se ejecuta el algoritmo de selección se retorna el elemento referenciado por el puntero o contador, mientras este último se aumenta en uno para hacer referencia al siguiente elemento del listado de posibles elecciones.

Si el elemento apuntado es el último de la lista entonces en la siguiente iteración al final del proceso de selección al incrementar en uno el contador se hará referencia al primer elemento de este listado [BSDO04]. Este algoritmo de selección asegura que todos los elementos de la lista eventualmente serán escogidos en algún proceso de elección.

Random

En esta estrategia no adaptativa también se mantiene un listado de ubicaciones y un puntero apuntando a algún elemento de esta lista, tal como el algoritmo Round-Robin explicado anteriormente, pero en este caso la elección del siguiente elemento referenciado se realiza de manera aleatoria [BSDO04].

LeastLoaded

Esta estrategia adaptativa de elección de recursos se basa en el envío de peticiones hacia una de las ubicaciones dentro del listado de posibles recursos disponibles del sistema hasta el instante en que un valor umbral de carga sea alcanzado. Una vez alcanzado este valor las siguientes peticiones son transferidas a la ubicación dentro del listado de éstas que presente menor valor de carga en el sistema [BSDO04].

LoadMinimum

Esta estrategia adaptativa calcula la carga promedio de los servidores del listado de recursos del sistema que contienen los objetos de Base de Datos a ser consultados. Si la carga en una ubicación particular es muy alta en relación a la carga promedio y además es mayor que la carga del servidor con menor carga del sistema ponderado por un porcentaje umbral de migración entonces todas las siguientes peticiones serán transferidas a la ubicación con menor carga del sistema [BSDO04].

Minimize Conflicts First (MCF)

MCF es un algoritmo greedy[5] ideado en función de minimizar el número de transacciones conflictivas asociadas a diferentes réplicas de servidores de Bases de Datos. El funcionamiento del algoritmo inicialmente consta en tratar de asignar cada transacción de la lista de trabajo a la réplica que presenta conflictos con dicha transacción.

En caso de no existir conflictos el algoritmo trata de balancear la carga entre las réplicas, asignando el trabajo al servidor que minimice la carga total del sistema, maximizando el paralelismo [ZuPe08].

Maximize Parallelism First (MPF)

El algoritmo MPF de selección de recursos prioriza el paralelismo entre transacciones como opción primaria de elección. Inicialmente trata de asignar las transacciones con el objetivo de mantener equilibrada la carga entre los servidores. Si existe más de una opción factible de selección, el algoritmo escogerá al servidor que trata de minimizar los conflictos que pueden ocurrir entre las transacciones.

Para comparar la carga entre servidores se utiliza un factor f, $0 < f < 1$, el cual permite establecer que dos servidores poseen carga similar si el cociente entre la carga de ambos servidores es mayor que f y menor que 1. Por ejemplo, un algoritmo MPF con factor $f = 0,5$ permite que la diferencia de carga entre dos servidores sea máximo un 50 %.

Cabe destacar que MCF es un caso especial del algoritmo MPF con factor f=0. [ZuPe08].

[5]Un algoritmo greedy (avaro) es un tipo de algoritmo de optimización que construye una solución en una o más etapas, y que en cada etapa escoge el mayor beneficio inmediato posible dentro de las opciones factibles [DaPV08].

2.2.2. Aplicaciones de Distribución de Consultas

Existen distintas aplicaciones (tanto comerciales, gratuitas o de código libre) que actúan como software intermediario en un clúster de computadores para brindar balanceo de carga al sistema. A continuación se listan algunas aplicaciones compatibles con un clúster de servidores de Bases de Datos MySQL.

MySQL-Proxy

MySQL-Proxy[6] es un programa que se sitúa entre el cliente y el servidor (o los servidores) MySQL. Es capaz de monitorear, analizar y transformar las consultas y comunicaciones que son enviadas a él.

La última revisión de MySQL-Proxy es la versión 0.8.1. Está programado en lenguaje C en su arquitectura general y utiliza scripts en lenguaje LUA[7] para especificar la secuencia de acciones a realizar ante cada consulta. Actualmente el programa cuenta con un amplio número de desarrolladores de la comunidad MySQL trabajando activamente en el proyecto.

Algunos de los usos comunes de MySQL-Proxy son:

- Distribución de carga.
- Recuperación de sistemas de Bases de Datos en caso de fallas.
- Análisis, filtro y modificación de consultas.

Shard-Query

Shard-Query[8] es un motor de consultas paralelamente distribuido. Consta de una clase escrita en lenguaje PHP cuya intención es hacer fácil el trabajo con un conjunto de datos particionado. Éste puede ser utilizado transparentemente e incluye un script en lenguaje LUA para MySQL-Proxy.

Las características claves de Shard-Query son las siguientes.

[6]Descripción completa de MySQL-Proxy en `http://dev.mysql.com/doc/refman/5.1/en/mysql-proxy.html`

[7]LUA Programming Language. Sitio web: `http://www.lua.org/`

[8]Descripción completa de Shard-Query en `http://code.google.com/p/shard-query/`

- Ruteo de consultas, es decir que las consultas son enviadas sólo al fragmento que efectivamente tiene los datos solicitados.
- Acceso en paralelo a las distintas fragmentaciones de los datos.
- Las agregaciones, uniones y filtros siempre son realizados a nivel de los fragmentos de los datos, lo cual distribuye completamente el trabajo total.
- Se hace uso del framework Gearman para agregar la capacidad de utilización de múltiples procesadores.

En la figura 2.3 se muestra un esquema del funcionamiento de Shard-Query.

2.3. Sistemas de Archivos en Red

Un sistema de archivos en red es cualquier sistema de archivos que permite compartir archivos, impresoras y otros recursos como un almacenamiento persistente en un conjunto de equipos conectados en red. Un clúster de computadores comparte recursos entre las máquinas, entre ellos al menos un directorio de archivos [IOWA11]. Se hace necesaria entonces para el sistema a implementar la implantación de un sistema de archivos en red en los distintos servidores que compondrán el clúster.

Varios protocolos han sido desarrollados para sistemas operativos específicos. En particular, las Bases de datos de Penta Analytics están instaladas sobre sistemas operativos basados en Linux. En estos ambientes el protocolo más utilizado y conocido es NFS [French11]

En Penta Analytics se ha implementado ya el uso de NFS como sistema de archivos distribuido, y existe documentación sobre cómo levantar este protocolo entre distintos servidores. El desempeño de NFS depende también del medio físico sobre el cual se interconectan los servidores, por lo que esto es un factor importante. Por ejemplo, puede utilizarse medios para Ethernet (100Mbs, 1Gbps, 10Gbps), o más sofisticados y de alta velocidad como InfiniBand [TaGi08].

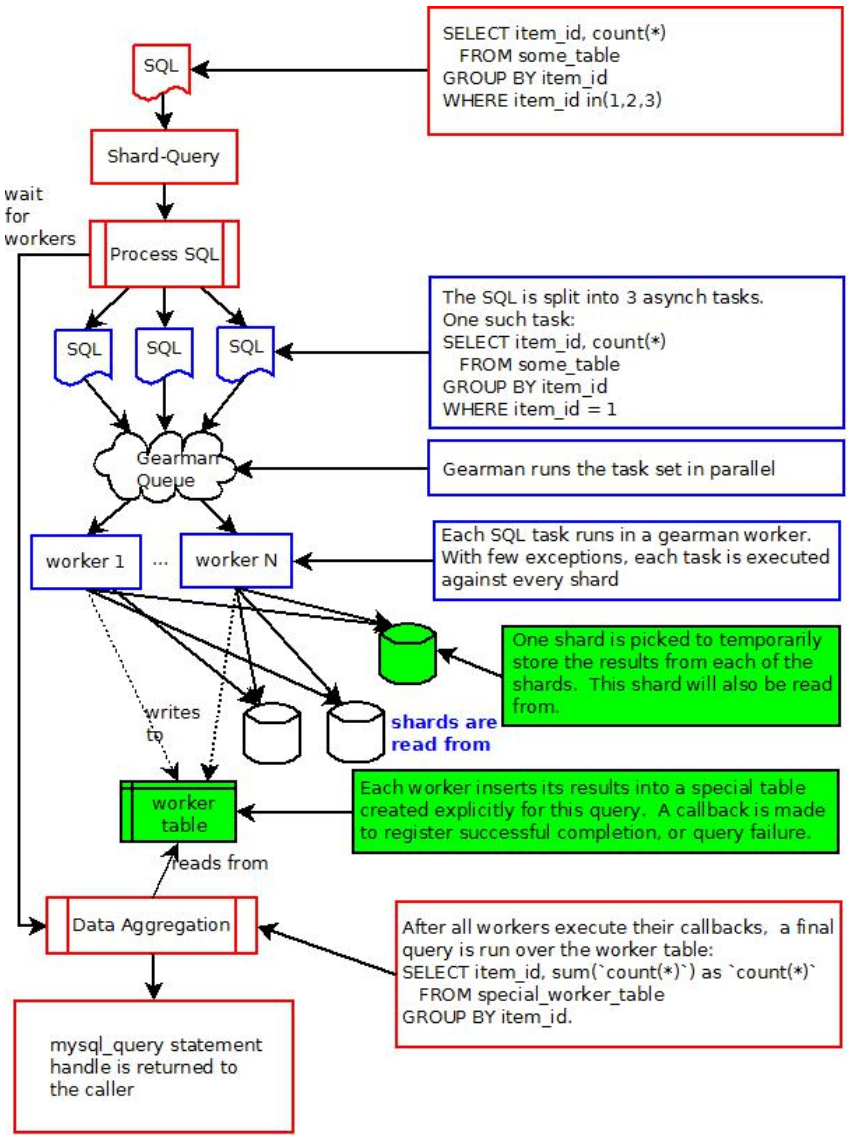

Figura 2.3: Esquema del funcionamiento de Shard-Query ante la ejecución de una consulta SQL. Fuente: [CoSha11]

2.4. Bases de Datos Infobright

Penta Analytics utiliza Infobright[9] como motor de Bases de Datos en sus servidores. Ésta posee un enfoque de almacenamiento y consultas orientado a columnas, es decir, los registros son guardados agrupando los registros por columnas, en contraste con el enfoque orientado a filas en donde los registros se almacenan agrupando por filas.

[9] `http://www.infobright.com`

2.4.1. Arquitectura

La arquitectura de Infobright posee varias capas, siendo las más trascendentes la capa de compresión de datos y la grilla de conocimiento. La primera de éstas comprime los datos agrupados en columnas, mejorando de esta forma el radio de compresión al realizar la tarea sobre el mismo tipo de datos. Cada 65.000 registros se genera un pack de datos con respecto a una columna y se guarda información relevante sobre éstos, como por ejemplo la suma de los registros, el promedio, el índice del primer y del último elemento, entre otros. Estos packs, en conjunto a otros datos relevantes, son resguardados en la grilla de conocimiento, la cual cumple la función de una verdadera biblioteca de información [Info08].

Un esquema de la arquitectura completa de Infobright se muestra en la figura 2.4.

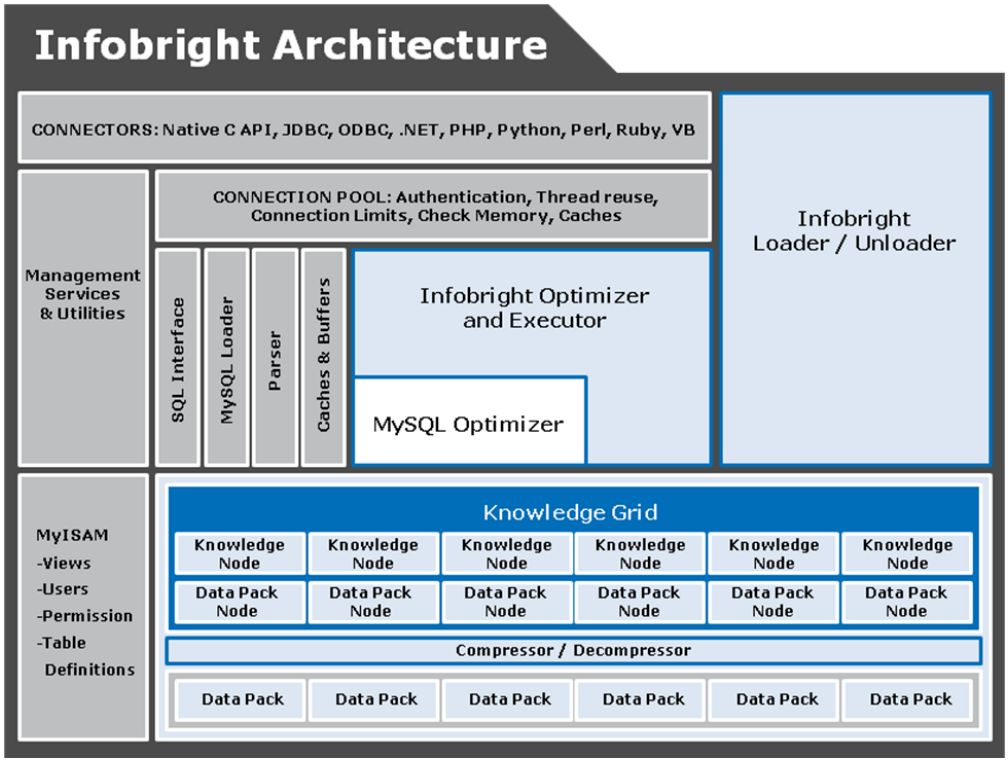

Figura 2.4: Esquema de la arquitectura de Infobright. Fuente: [Info08]

Este motor de Base de Datos cuenta con dos versiones para el producto, las cuales son *Infobright Enterprise Edition (IEE)* e *Infobright Community Edition (ICE)*. El resumen de la diferencias más importantes entre estas versiones es el siguiente [IEEICE11].

Suscripción por uso

La versión IEE exige el pago de una suscripción, la cual permite utilizar una licencia para su uso; en cambio la versión ICE es gratuita y su licencia es la GPL v2[10].

Soporte a operaciones DML *(Data Manipulation Language)*

La versión IEE brinda soporte completo a las operaciones SQL de tipo DML, es decir, permite las instrucciones SQL INSERT, UPDATE y DELETE. La versión ICE no brinda soporte a estas instrucciones y la carga de datos en este caso se realiza a través del denominado *Infobright Loader*, utilizando el comando LOAD DATA INFILE. Cualquier cambio en los datos almacenados en una versión ICE requiere de una nueva subida completa de los datos por cada tabla involucrada.

Soporte a comandos ALTER TABLE/RENAME TABLE

IEE brinda soporte a estos comandos, los cuales permiten añadir, cambiar o borrar columnas y atributos. La versión ICE sólo brinda soporte al comando RENAME TABLE.

Soporte a replicación basada en statements

La replicación de MySQL basada en statements (en donde se intercepta cada consulta SQL y se distribuye a todo el clúster de servidores) es soportada oficialmente desde la versión 3.5 en IEE. Para ICE no existe soporte de replicación.

Soporte a tablas temporales

IEE brinda soporte completo a los comandos SQL de creación de tablas temporales, mientras ICE no brinda soporte a este tipo de tablas.

[10] `http://www.gnu.org/licenses/gpl-2.0.html`

Capítulo 3

Diseño e Implementación

En este capítulo se plasman los detalles referentes al diseño y la implementación del presente tema de memoria. En primer lugar se describe el resultado de la investigación realizada[1], el cual impacta directamente en el diseño y en la arquitectura general del sistema. Luego se describen los requisitos funcionales y no funcionales determinados para la solución y finalmente se explican los detalles de cada uno de los componentes, tanto replicación como distribución de consultas, que conforman la solución programada.

3.1. Restricciones, decisiones preliminares de diseño y alcances

3.1.1. Restricciones

Se decidió trabajar con un esquema donde una Bases de Datos IEE es la *maestra* y varias ICE actúan como *esclavas*. Esta decisión tiene como raíz dos argumentos: el primero es porque ICE cumple a cabalidad el rol precisado para una Base de Datos de tipo analítica, y el segundo se basa en no encarecer la solución de alta disponibilidad, pues los costos de suscripción por uso de Infobright IEE superan los miles de dólares.

Cabe destacar que la replicación nativa que ofrece Infobright en su producto empresarial requiere que todas las Bases de Datos secundarias sean de tipo IEE (es decir, un clúster de Bases de Datos IEE), lo cual imposibilitó, por el mismo argumento anterior, la utilización de este tipo de replicación.

Con este esquema de trabajo (una IEE *maestra* y varias ICE *esclavas*) las decisiones sobre

[1]Para mayor información véase el capítulo 2.

los tipos de replicación a utilizar fueron simplificadas. Como ICE no soporta operaciones SQL de tipo DML y como no existe forma de propagar los cambios sin realizar previamente operaciones de *dump*[2] en la Base de Datos principal, entonces se decidió realizar replicación de tipo Copia Primaria, Asíncrona y via Snapshot.

Por otro lado, de las aplicaciones investigadas Tungsten no provee el tipo de replicación definido para este trabajo y alguna modificación a éste representa un cambio sustancial en la arquitectura de ésta, el cual se estimó que implicaba un tiempo mayor al considerado para este trabajo, por lo cual fue descartada. Gearman, a pesar de ser un framework con múltiples usos, nativamente no provee soporte para ejecutar trabajos de forma remota en un servidor específico, la cual es una característica indispensable para la solución a implementar; al igual que en el caso anterior, la modificación de Gearman se estimó que implicaba un tiempo de trabajo mayor al establecido para esta memoria. Basándose en los argumentos anteriores es que se optó diseñar una aplicación de replicación ad hoc a Infobright.

3.1.2. Decisiones de diseño

El lenguaje de programación escogido para la programación de la solución fue Java, el cual es un lenguaje robusto, con vasta documentación y que posee soporte para la ejecución remota de procedimientos. Java RMI es una tecnología presente desde hace más de 10 años y existe un gran número de aplicaciones empresariales que utilizan RMI para la ejecución remota de procedimientos.

Se espera que sean el sistema de replicación en conjunto con el servidor esclavo los que establezcan que una actualización ha sido finalizada, pues el sincronizar a todos los servidores para decidir si una actualización fue exitosa puede tomar bastante tiempo (por la carga de datos en los servidores *esclavos*) y basta que un servidor *esclavo* falle en la carga para que todo el proceso de actualización no sea exitoso. Por otro lado se decidió que estas actualizaciones no deben ser gatilladas por un evento específico, sino que se produzcan a intervalos de tiempo determinados. Considerando estos argumentos es que se especifica que el tipo de replicación debe ser además Periódica y con Terminación No Votada.

[2]Un *dump* de una tabla es un archivo o serie de archivos que contienen la estructura de la tabla involucrada, en conjunto con los datos de ésta. Usualmente estos archivos son conformados por una lista de instrucciones SQL.

De las aplicaciones de distribución de carga, la que ofrece mayor flexibilidad para modificar, inyectar y rutear consultas es MySQL-Proxy. Ésta es compatible con cualquier versión de MySQL superior a la 5.0. Infobright, como se mencionó en el capítulo 1, posee alta similitud con MySQL al ser éste una eventual máscara para el motor de la Base de Datos. La flexibilidad de programar scripts en LUA se traduce en un mayor control sobre la distribución de las consultas, sin caer en pérdidas sustanciales de tiempo. Por estos motivos es que se incorporó a MySQL-Proxy como el sistema que brinda distribución de consultas.

El algoritmo de elección de servidor de backend escogido para este trabajo es Round-Robin, con la modificación necesaria para que se distinga entre una consulta de actualización y una consulta de lectura de datos. Este algoritmo entrega una simple pero efectiva estrategia de balanceo de carga y en su ejecución no se precisa de una lógica que extraiga información adicional a la que MySQL-Proxy le pueda proporcionar desde sus variables internas; por ejemplo, no requiere obtener la carga de los servidores en su proceso de selección.

Se realizó una exploración de diseños para la aplicación de replicación, buscando determinar cuál enfoque impacta menos en el rendimiento del sistema. Se estudió en particular cómo determinar el estado de actualización de una tabla y cuál es el grado de interrelación que deben tener el software de replicación y el de distribución de consultas.

El grado de acoplamiento de los sistemas es indispensable que sea bajo, para que en caso de falla de alguno de éstos no implique en una interrupción también de los servicios del otro componente. Por tanto, el estado de actualización de cada tabla del sistema debe ser posible de obtener de forma independiente por cada aplicación.

El primer enfoque de diseño se basó en almacenar el estado de modificación de una tabla, el cual puede ser obtenido desde la tabla *TABLES* de la Base de Datos *INFORMATION_SCHEMA* de cada servidor del clúster. La lógica en este caso pasaba por almacenar la fecha de modificación de los datos de la tabla *maestra* cargados en los servidores esclavos; es decir, cada vez que se cargan datos en un servidor *esclavo* se almacena la fecha de modificación de la tabla maestra de origen.

Entonces, la lógica del sistema de replicación pasa por obtener la fecha de modificación de cada tabla *maestra* y compararla con la fecha almacenada en el sistema de replicación para cada tabla *esclava*. En caso de que la fecha de modificación de la tabla *maestra* fuese

más reciente que la almacenada en el sistema, se gatilla el proceso de descarga y carga de datos por cada tabla desactualizada del sistema.

Este diseño también impacta en el sistema de distribución de carga, pues para establecer cuál servidor *esclavo* podría responder adecuadamente a una consulta se debe ejecutar la misma lógica que utiliza el sistema de replicación; es decir, el sistema debe acceder al lugar de almacenamiento del estado de las tablas del clúster y realizar una operación de comparación de fechas para cada tabla esclava.

El otro diseño estudiado se basa en almacenar en una tabla (llamada *tabla de replicación*) el estado de actualización, sin consultar la fecha de modificación como en el caso del diseño anterior, sino que almacenando una variable (el *estado de actualización*) que refleje si una tabla del clúster está actualizada o no. Se asume que una tabla de un servidor *maestro* siempre estará actualizada.

Ambos sistemas (replicación y distribución de consultas) en este sistema acceden y modifican los valores de esta tabla. El software de distribución es el encargado de marcar las tablas *esclavas* como desactualizadas cada vez que se realiza una operación de actualización, y el sistema de replicación vuelve a marcar las tablas como actualizadas una vez cargados correctamente los datos y si es que no han vuelto a ser modificados en el maestro. Lo anterior es trascendental para el sistema, pues una incorrecta modificación en la tabla (marcar una tabla como actualizada, siendo que efectivamente no lo está) implica que el sistema queda en un estado no consistente.

De los dos diseños propuestos se tomó la decisión de implementar el segundo de éstos; es decir, utilizando un estado de actualización en la tabla de replicación y no realizando una lógica sobre las fechas de actualización de las tablas del sistema. Este diseño ejecuta menos operaciones de comparación para determinar el estado de actualización, lo cual se traduce en una lógica más simple y precisa menos tiempo en su ejecución.

Este último aspecto es crítico para el sistema de distribución de consultas, pues se espera que la lógica que determina el estado de actualización impacte mínimamente en el tiempo de respuesta de la consulta.

Una última decisión importante tiene relación con la concurrencia en las operaciones de modificación del estado de actualización del sistema, lo cual es crítico para mantener

la consistencia total del sistema. Por tanto, cualquier modificación que se desee realizar en el estado mencionado debe considerar que otras operaciones no interfieran en una correcta operación de la actualización y no impliquen una pérdida de consistencia. Gracias a las operaciones de transacciones y bloqueo de tablas, propias de las Bases de Datos, se puede manejar de correcta forma la concurrencia, pero se debe realizar una programación cuidadosa para aprovechar este tipo de control.

3.2. Requisitos

En esta sección se presentan los requisitos más importantes de la solución a desarrollar, en base a las restricciones y decisiones de diseño establecidas. A continuación una breve descripción de cada uno de ellos.

1. Funcionales

    1.1. La solución de replicación debe mantener el estado de actualización de cada servidor del *clúster*.

    1.2. La solución de replicación debe distribuir los cambios a nivel de tablas desde la Base de Datos *maestra* a las Bases de Datos *esclavas*.

    1.3. La solución de distribución de consultas debe distinguir entre las consultas que escriben datos y las que sólo leen datos.

    1.4. La solución de distribución de consultas debe escoger el servidor al cual enviar la consulta en base a algún algoritmo cuyo objetivo sea mejorar el desempeño general del sistema.

    1.5. La solución de distribución de consultas debe obtener información sobre el estado de actualización del servidor.

2. Restricciones

    2.1. La solución debe operar sobre un clúster de Bases de Datos analíticas Infobright.

    2.2. Los servidores Infobright Community Edition no soportan operaciones de actualización sobre los registros de una tabla.

2.3. La solución, por restricciones de licenciamiento y funcionalidades permitidas, debe considerar una Base de Datos Infobright Enterprise Edition como Base de Datos *maestra* y una o más Bases de Datos Infobright Community Edition como Bases de Datos *esclavas*.

2.4. La solución debe ser compatible con el servidor de Bases de Datos MySQL, dada la estructura similar entre este motor e Infobright [Info10].

2.5. La solución debe considerar la transmisión y carga eficiente de datos de gran volumen entre distintos servidores de Bases de Datos.

3.3. Arquitectura de la solución

En este apartado se mencionan aspectos generales relevantes de la solución desarrollada, como las interacciones del sistema o las redes de conexión, con el fin de aclarar a grandes rasgos el funcionamiento y el panorama general del sistema.

3.3.1. Redes de conexión

En la figura 3.1 se muestra un esquema de la configuración de servidores, equipos, firewalls, conexiones y cómo se integra el sistema de alta disponibilidad en este escenario.

Para disminuir la complejidad del transporte de datos en red se escoge montar un sistema de archivos en red local, en particular el protocolo NFS, en cada uno de los servidores pertenecientes al clúster; de esta forma se hace abstracción de la ubicación de los directorios en cada servidor y se delega al sistema de archivos la complejidad de transmitir los datos de manera eficiente. En Penta Analytics el protocolo NFS ya se había implementado con anterioridad, por tanto se aprovechó la documentación y manuales existentes para proceder a su instalación en los distintos servidores.

3.3.2. Arquitectura General del Sistema

En la figura 3.2 se aprecia un esquema general del sistema de replicación y distribución de consultas.

Los sistemas que componen la solución (destacados en la figura 3.2 con (1) y (2)) fueron diseñados de forma tal que cada uno de éstos no dependa del otro para entregar sus funciones,

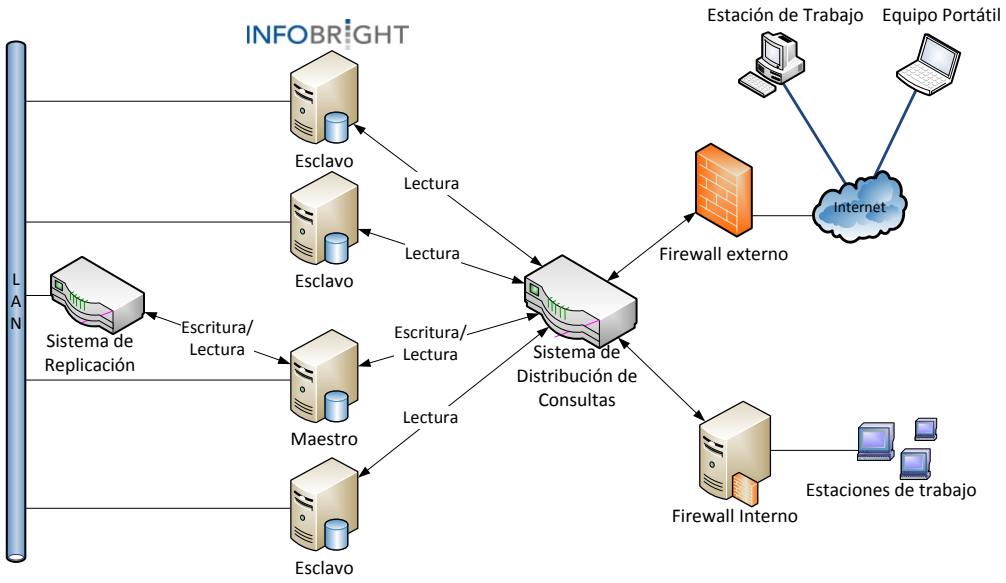

Figura 3.1: Esquema de redes, conexiones e interacción con la solución implementada. Fuente: elaboración propia.

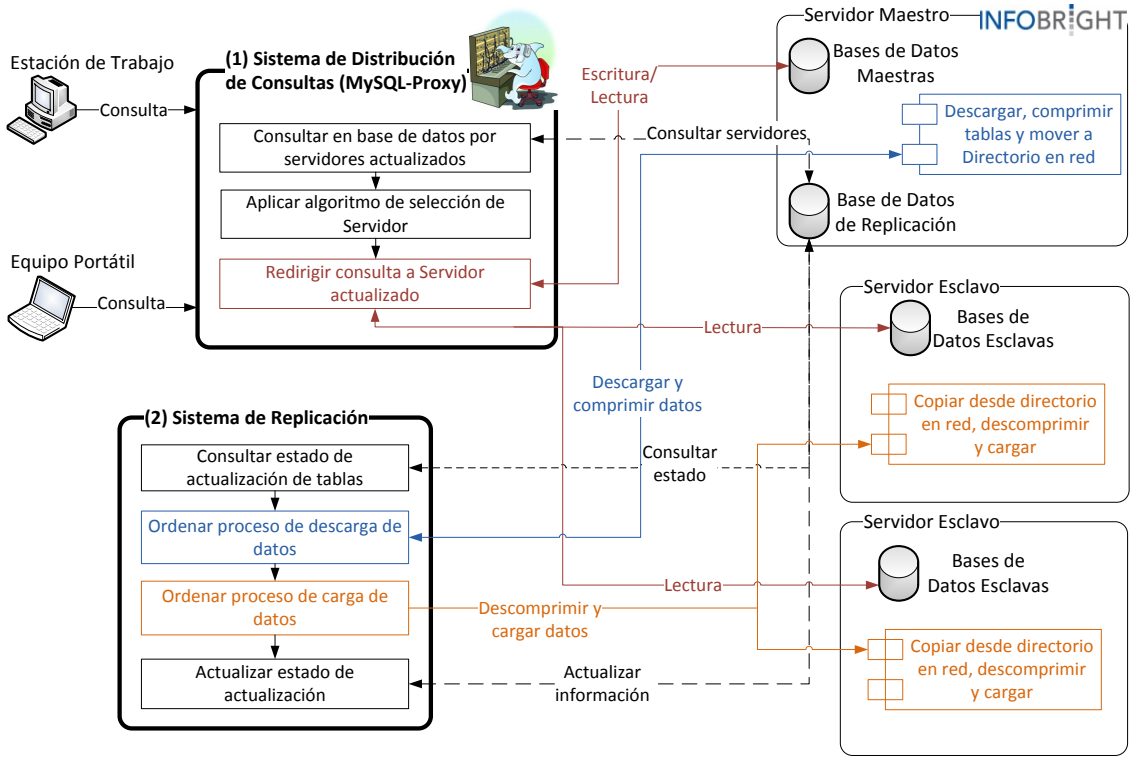

Figura 3.2: Esquema general de la solución implementada. Fuente: elaboración propia

como se especificó en las decisiones preliminares de diseño. El único punto en común que comparten ambos es la tabla de replicación alojada en el servidor *maestro*.

El sistema de distribución de carga (1) brinda sus servicios aún cuando el sistema de replicación (2) no se encuentre funcionando. El comportamiento esperado en este caso es que las Bases de Datos *esclavas* una a una sean marcadas como desactualizadas, en la medida que las actualizaciones van teniendo lugar en el servidor *maestro*.

Las responsabilidades definidas para cada sistema son las siguientes.

Distribución de Carga

- Redirigir las consultas a servidores esclavos en caso de ser necesario.
- Marcar tablas como desactualizadas ante operaciones de actualización o modificación.

Replicación

- Descargar los datos de servidor principal y cargarlos en los servidores *esclavos* desactualizados.
- Marcar tablas como actualizadas luego de un proceso de replicación exitoso.

3.3.3. Tabla de replicación

Se estableció que el estado de actualización de cada tabla del sistema sea almacenacedo en una tabla del servidor *maestro* de datos. De esta forma se relega en el motor de Base de Datos y en una minuciosa programación los temas críticos de concurrencia.

La información mínima que el sistema debe considerar sobre cada una de las tablas de la Base de Datos del sistema es la siguiente.

- Nombre de la Base de Datos de la tabla. Puede tener cualquier valor alfanumérico.
- Nombre de la tabla. Puede tener cualquier valor alfanumérico.
- Dirección ip o nombre del servidor. Sólo debe poseer valores numéricos y puntos, como por ejemplo: “192.168.124.132”.
- Estado de actualización (actualizada, desactualizada o actualizando).
- Fecha de modificación.

Dada que los eventos de actualización se llevan a cabo siempre en primer lugar en el servidor *maestro*, se considera entonces que los datos almacenados en este servidor, en el contexto general de la solución, siempre tendrán el status de actualizados. Entonces, en la tabla de replicación su estado de actualización será inmutable, pues no hay forma que cambie de estado a desactualizado o actualizando; por tanto se puede simplificar el diseño optando por no almacenar el estado de actualización del servidor *maestro* de datos.

El campo que hace referencia al estado de actualización es crítico para el sistema, pues es en base a éste que tanto el sistema de replicación como el de distribución de consultas tomarán las decisiones para establecer cuándo una tabla está actualizada. Además, se requiere que las operaciones de cambio de estado sean atómicas[3] en su ejecución.

Se estableció que los estados de actualización para una tabla del sistema y sus significados son los siguientes.

Estado 1 La tabla se encuentra actualizada en este servidor.

Estado 0 La tabla de este servidor se encuentra en proceso de copia de datos desde sistema de archivos en red y posterior carga de datos.

Estado -1 La tabla de este servidor se encuentra desactualizada.

Estado -2 La tabla de este servidor se encuentra desactualizada y no se ha ejecutado aún alguna descarga de datos.

El estado -2 fue creado para optimizar la cantidad de *dumps* realizados; si alguna tabla posee este estado significa que aún no se ha ejecutado un dump exitoso, pero es posible que esté en ese proceso; con lo anterior se pretende diferenciar entre una tabla que está desactualizada pero con proceso de dump ejecutándose y una tabla desactualizada sin este proceso. Para pasar de desactualizada (estado -1) a actualizándose (estado 0) es preciso pasar por el estado intermedio (estado -2).

[3]Se dice que una operación es atómica cuando es imposible encontrar para cualquier otro componente del sistema al cual pertenece algún paso intermedio. Si esta operación consiste en un cambio de estado, entonces el cambio tiene éxito cuando todas las operaciones que componen este cambio son exitosas en su completitud, o en caso contrario no se efectúa la operación.

Si el sistema detecta que alguna tabla tiene el estado 0, esto significa que se realizó una descarga de datos con éxito, pero falló la carga de datos. Por tanto, no se necesita realizar una nueva descarga y se puede continuar con la carga de datos respectiva en los servidores *esclavos*.

La figura 3.3 ilustra las transiciones válidas para este sistema según la lógica diseñada para los sistemas de replicación y distribución de consulta. Se indica en las flechas de transición el sistema encargado de ejecutar la operación atómica de cambio de estado.

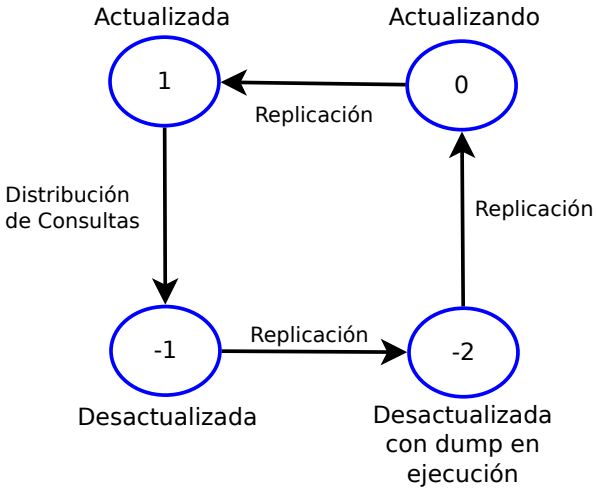

Figura 3.3: Diagrama de estados y transiciones del estado de actualización de las tablas. Fuente: elaboración propia.

Para garantizar la atomicidad de las operaciones de cambio de estado se estableció el uso de transacciones SQL. Las transacciones son un conjunto de instrucciones SQL tales que el sistema gestor de Bases de Datos garantiza la atomicidad y consistencia de éstas. Con ésto se puede encapsular una consulta de actualización al servidor *maestro* de datos junto a una operación de actualización de los datos, de tal forma que si alguna de éstas falla ninguna de ellas sea efectivamente ejecutada.

El contenido de la tabla de replicación para este trabajo fue agregado manualmente. No existe un agente o programa que compruebe que una tabla agregada en la tabla de replicación efectivamente exista tanto en el *maestro* como en los *esclavos*.

Para agregar un servidor como *esclavo* del clúster basta con agregar en la tabla de replicación la ip del servidor por cada tabla de las Bases de Datos del *maestro*.

Considerando esta información, fue creada la tabla de replicación, cuya estructura aparece

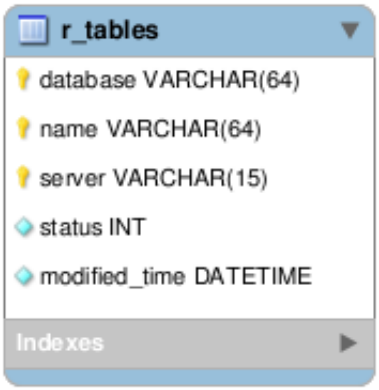

Figura 3.4: Estructura de tabla mantenedora de estados de actualización. Fuente: elaboración propia

en la figura 3.4.

Establecida la estructura de la tabla se procedió al diseño del sistema de replicación.

3.4. Sistema de Replicación

La solución de replicación implementada consta de dos programas: La aplicación principal, que determina qué elementos están actualizados o desactualizados y se comunica con las instancias de la aplicación secundaria, encargada de ejecutar la carga o descarga de datos en cada servidor según corresponda.

El programa principal lee los datos desde la tabla de replicación para obtener los elementos desactualizados y procede a ejecutar la lógica de actualización. Por otro lado, el programa secundario se encarga de descargar los datos, mover los *dumps* al directorio en red compartido y de cargar los datos en el servidor correspondiente. En la figura 3.5 se ilustra un esquema del proceso que se realiza en el sistema.

3.4.1. Replication.jar

La aplicación principal se encuentra empaquetada en el archivo **Replication.jar** y consta de dos *packages*, **cl.analytics.replication** y **cl.analytics.utils**, que contienen las clases que componen a la aplicación. En la figura 3.6 se ilustra el diagrama de clases para la aplicación Replication.jar.

Esta aplicación precisa de librerías externas para su ejecución, las cuales son las siguientes.

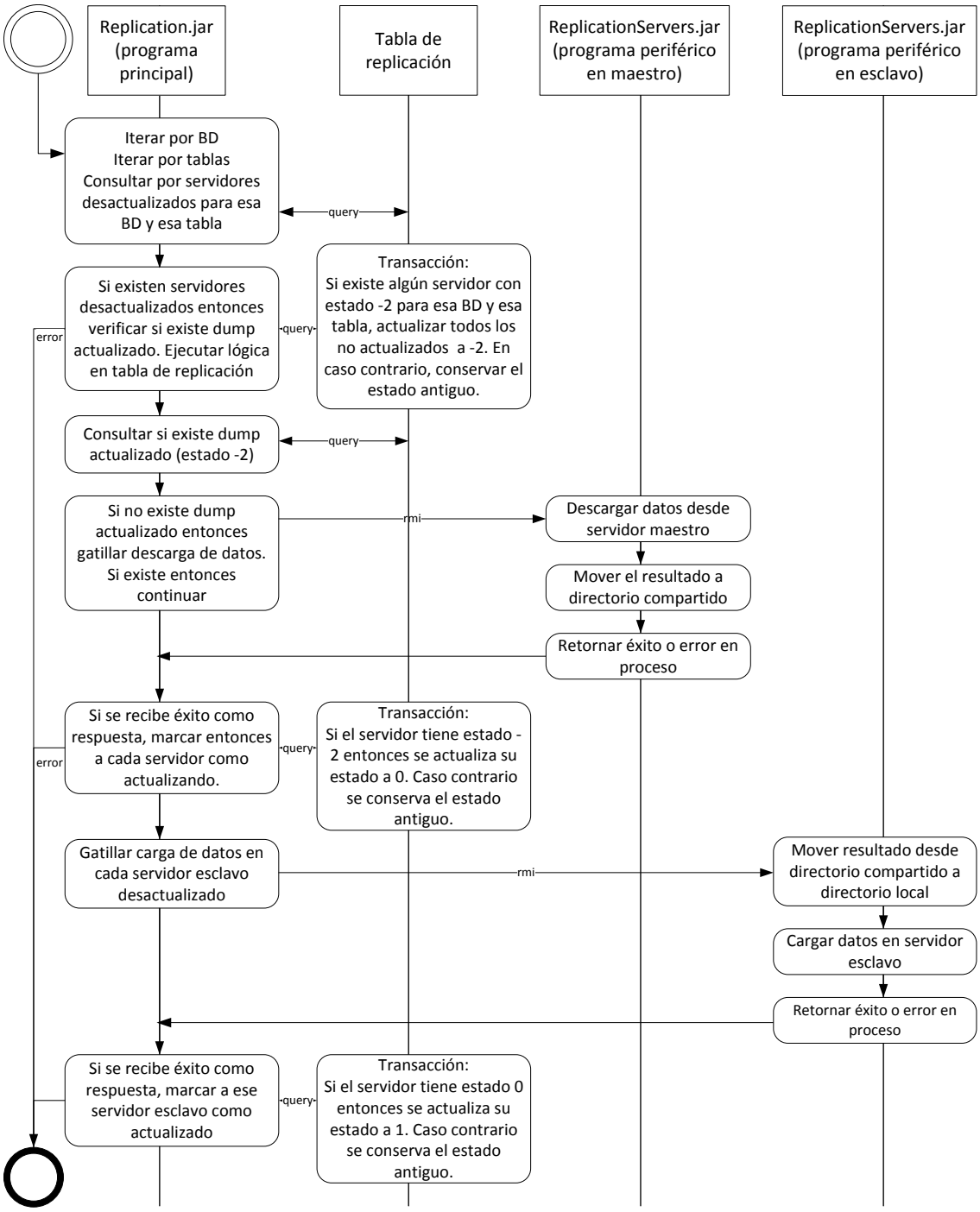

Figura 3.5: Esquema del proceso ejecutado por la solución de replicación. Fuente: elaboración propia

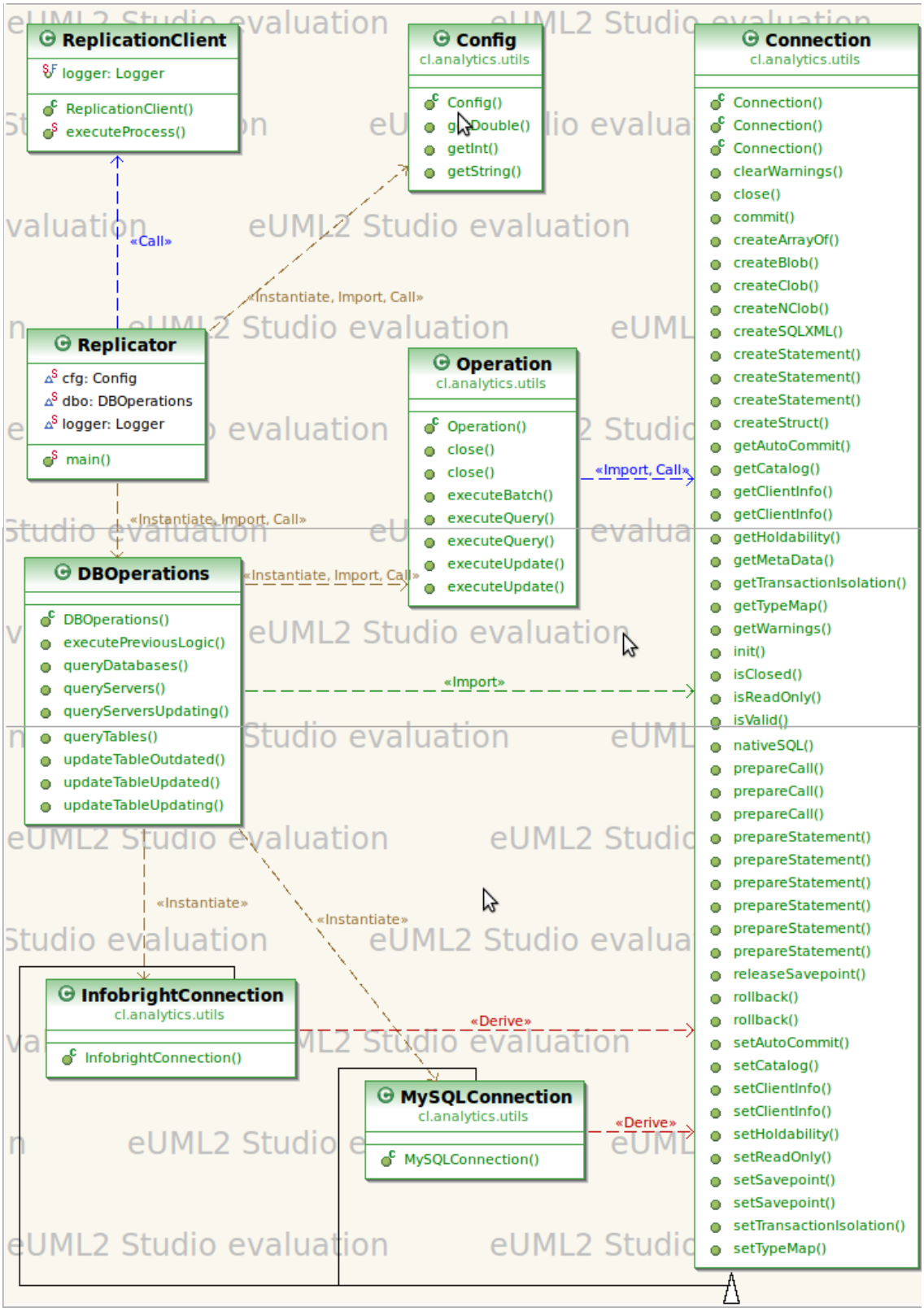

Figura 3.6: Diagrama de clases de la aplicación Replication.jar. Fuente: elaboración propia.

Commands.jar Librería que contiene la cabecera de los métodos que pueden ser ejecutados por la aplicación secundaria.

log4j-1.2.16.jar Librería para crear logs sobre los pasos ejecutados por el programa[4].

mysql-connector-java-5.1.8-bin.jar Librería para crear conexiones y ejecutar comandos en una Base de Datos MySQL[5].

El programa principal se encuentra en la clase **Replicator.class**, dentro del package **cl.analytics.replication**, y no necesita parámetros de ejecución pero sí precisa de dos archivos, llamados **replication.properties** y **log4j.properties**, los cuales deben estar ubicados en el directorio en donde se está ejecutando el programa. El primero de éstos contiene parámetros de configuración para la aplicación, tales como el nombre de usuario, la dirección ip del servidor principal, etc., mientras el segundo archivo contiene los datos sobre el nivel de profundidad en que se informan las acciones que el sistema ejecuta.

En líneas generales el programa principal realiza el siguiente proceso.

- Obtener todas las Bases de Datos del sistema (desde la tabla de replicación), almacenarlas en una lista e iterar sobre éstas.
- Por cada Base de Datos obtenida, consultar a la tabla de replicación por todas las tablas pertenecientes a esta Base, e iterar sobre cada una de éstas.
- Por cada tabla obtenida, consultar por los servidores esclavos asociados a ésta que tengan estado desactualizado e iterar sobre éstos.
    - En caso de que el listado de servidores sea vacío, omitir esta tabla y continuar con la siguiente del listado.
    - En caso de no ser vacío, se ejecuta una lógica que verifica si existe algún *dump* actualizado de los datos (estado -2 en algún elemento del listado).
        - En caso de existir, se omite el proceso de dump en el servidor *maestro*.

[4]Esta librería se puede descargar desde `http://www.apache.org/dyn/closer.cgi/logging/log4j/1.2.16/apache-log4j-1.2.16.tar.gz`

[5]Esta librería se puede descargar desde `http://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.18.tar.gz/from/http://linorg.usp.br/mysql`

- Si no existe alguno, se procede a realizar el proceso de *dump* en el servidor *maestro*.
- Si en los pasos anteriores no se obtuvieron errores, entonces se procede a la carga de los datos en cada servidor *esclavo* del listado.
- Si la carga de datos fue exitosa, se ejecuta una transacción SQL en donde se verifica si el estado del servidor que se está actualizando es 0. En caso de ser así, se marca ese registro como actualizado (estado 1), en caso contrario se mantiene el estado anterior.

- En caso de falla en algún proceso, se captura la excepción, se omite el elemento que falló (ya sea una Base de Datos, una tabla o un servidor) y se continúa con el siguiente elemento del listado respectivo.

- Finalizado el proceso completo, el sistema espera durante un tiempo configurable para continuar con otra iteración.

Las otras clases que componen el package **cl.analytics.replication** son **DBOperations.class** y **ReplicationClient.class**. La primera de éstas se utiliza para las comunicaciones y consultas a la Base de Datos, mientras las segunda se encarga de establecer la comunicación con los programas periféricos instalados en otros servidores mediante RMI.

Las consultas a Bases de Datos que implican modificaciones en la tabla de replicación se realizan mediante transacciones SQL y operaciones de LOCK a nivel de tabla, con el fin de asegurar que durante la transición de un estado a otro no exista la posibilidad de que un tercero haga una actualización y pueda corromper la consistencia de la tabla de replicación. Un ejemplo del tipo de transacciones que ejecuta la clase **DBOperations.class** es la siguiente.

```
START TRANSACTION;
LOCK TABLE Replication.r_tables as rt WRITE;
SET @old_status = (SELECT rt.status
FROM Replication.r_tables as rt
WHERE rt.database='cliente'
AND rt.name='cliente_bdm'
AND rt.server='192.168.2.3' );
UPDATE Replication.r_tables as rt
```

```
SET rt.status = (SELECT CASE
WHEN @old_status = 0
THEN 1
ELSE @old_status END),
  modified_time = NOW()
WHERE rt.database='cliente'
AND rt.name='cliente_bdm'
AND rt.server='192.168.2.3';
UNLOCK TABLE;COMMIT;
```

La invocación remota de métodos, tarea que realiza la clase **ReplicationClient.class** requiere de cuatro parámetros para su ejecución: el tipo de operación (descarga o carga de datos), la dirección ip del servidor en el cual ejecutar, el nombre de la Base de Datos y el nombre de la tabla. En caso de que estas operaciones sean exitosas el valor de retorno es cero.

El otro package componente del archivo **Replication.jar** (**cl.analytics.utils**) contiene clases que apoyan la labor efectuada por el otro package. El siguiente listado describe cada una de estas clases y su funcionalidad.

Config.class Permite leer datos desde un archivo de configuración.

Connection.class Gestiona las conexiones con Bases de Datos.

InfobrightConnection.class Extiende a Connection, crea una instancia de gestión de conexión específica para Bases de Datos Infobright.

MySQLConnection.class Extiende a Connection, crea una instancia de gestión de conexión específica para Bases de Datos MySQL.

Operation.class Permite ejecutar consultas e instrucciones generales en una Bases de Datos.

UsefulFunctions.class Contiene algunas funciones útiles; en particular se utiliza la función *executeCommandLine*, que permite ejecutar mediante línea de comandos una instrucción.

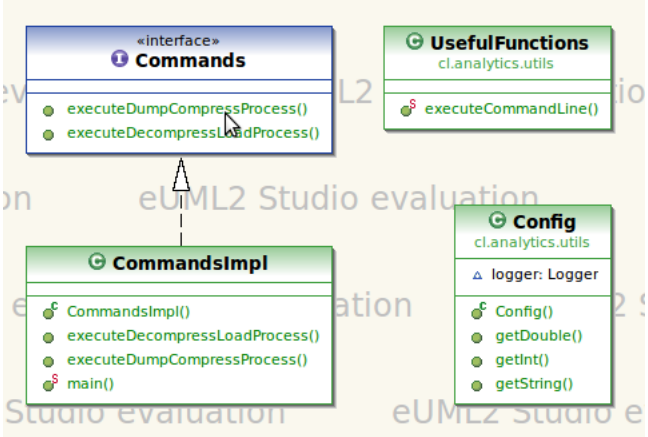

Figura 3.7: Diagrama de clases de la aplicación ReplicationServers.jar. Fuente: elaboración propia.

3.4.2. ReplicationServers.jar

La aplicación periférica que se instala en cada servidor del clúster está empaquetada en el archivo **ReplicationServers.jar** y consta de dos packages, **cl.analytics.replication.server** y **cl.analytics.utils**. En la figura 3.7 se muestra un diagrama de clases de la aplicación ReplicationServers.jar.

Esta aplicación sólo utiiliza una librería externa, **log4j-1.2.16.jar** y utiliza dos archivos de configuración, **log4j.properties** (que cumple la misma función indicada en la sección 3.4.1) y

En este caso el package *utils* es un versión simplificada del mismo package empaquetado en el archivo **Replication.jar**. Contiene tan sólo dos clases de las especificadas en la tabla **??**, **Config.class** y **UsefulFunctions.class**, que cumplen las mismas funcionalidades descritas allí.

El package **cl.analytics.replication.server** contiene dos clases, **Commands.class** y **CommandsImpl.class**. La primera de éstas es una interfaz que extiende a **java.rmi.Remote** para cumplir con el estándar de RMI y declara los métodos que pueden ser invocados remotamente. La segunda clase implementa a la primera, definiendo con claridad qué acciones ejecuta cada método. A continuación se muestra descripción de estos métodos.

executeDumpCompressProcess Recibe como parámetro una Base de Datos, una tabla y hace llamado por línea de comandos al script **dump_db_ib.sh** para generar un *dump*

de esta tabla en el directorio */var/tmp*. Si esta operación tiene éxito entonces mueve este *dump* al directorio en red compartido mediante el comando *rsync*. Retorna 0 en caso de que el proceso completo sea exitoso.

executeDecompressLoadProcess Recibe como parámetro una Base de Datos, una tabla y mueve desde directorio en red compartido el *dump* al directorio local */var/tmp* mediante *rsync*. Posteriormente ejecuta el script **recreate_table.sh** para recrear la tabla en el servidor local y luego ejecuta el script **restore.sh** para cargar los datos en la Base de Datos correspondiente. Retorna 0 en caso de que el proceso completo sea exitoso.

Creación de Commands.jar

La aplicación principal necesita de la librería **Commands.jar** para su funcionamiento. Ésta se crea a partir de la clase **Commands.class** y gracias a ella es que el programa conoce los métodos que pueden ser invocados por esta clase.

Para crear esta librería en el directorio */var/tmp* bastó con ejecutar desde una terminal el siguiente comando.

```
jar cvf /var/tmp/Commands.jar cl/analytics/replication/server/Commands.class
```

Scripts de descarga y carga de datos

La tarea de cargar y descargar los datos desde los servidores de datos la realizan scripts programados por los desarrolladores de Penta Analytics, los cuales fueron modificados para adaptar su funcionamiento con el sistema de replicación. Estos scripts son incluidos en una carpeta llamada */scripts*, en el mismo directorio donde se encuentra el archivo principal. A continuación se describen los scripts utilizados y su funcionamiento.

dump_db_ib.sh Este script recibe como parámetros el nombre de la Base de Datos, la tabla, la contraseña de conexión del usuario root y crea, en caso de no existir, o borra en caso de existir, el directorio */var/tmp/(nombre de la base de datos)/(nombre de la tabla)*. Luego prosigue creando un archivo sql con la estructura de la tabla (**schema_no_charset.sql**) y finalmente ejecuta la descarga de datos, comprimiendo los resultados con la aplicación lzma.

recreate_table.sh Este script recibe como parámetro el nombre de la Base de Datos, la tabla y recrea la estructura de la tabla en el servidor local, utilizando el archivo *schema_no_charset.sql*.

Para que el proceso de replicación funcione sin problemas se estableció como requisito que al menos el nombre de la Base de Datos esté creada en los servidores *esclavos*.

3.5. Sistema de Distribución de Consultas

Este sistema está basado en MySQL-Proxy, el cual redirige las conexiones entrantes a algún servidor integrante del clúster de Bases de Datos Infobright que esté actualizado y se encuentre disponible para atender consultas.

Se debe configurar mediante un archivo de texto algunas propiedades de MySQL-Proxy antes de iniciar su ejecución. Por ejemplo, es necesario describir cuáles son los servidores que serán de sólo lectura o sólo escritura, si existe algún archivo de log que almacene la traza de los eventos de la aplicación, etc. La figura 3.8 muestra un ejemplo del archivo de configuración utilizado por la aplicación, llamado **mysql-proxy.config**.

```
1 [mysql-proxy]
2 log-file=/var/tmp/mysql-proxy.log
3 log-level=debug
4 admin-username=ronald
5 admin-password=864927
6 admin-lua-script=
7 proxy-backend-addresses=192.168.2.219:5029
8 proxy-read-only-backend-addresses
     =192.168.2.218:5029,192.168.2.235:5029,192.168.2.237:5029
9 proxy-lua-script=/home/ronald/Desktop/mi-proxy.lua
```

Figura 3.8: Ejemplo de archivo de configuración de MySQL-Proxy. Fuente: archivo original.

Mediante el proxy se interceptan las consultas que son enviadas a las Bases de Datos y ejecutar la lógica que escoja un servidor adecuado para responder ante ésta. Además de lo anterior, en particular en este trabajo se requiere identificar cuáles son las tablas involucradas en el proceso, en caso de reconocer una instrucción SQL de modificación, borrado o actualización, para actualizar en la tabla de replicación el estado de las tablas esclavas. Lo

anterior debe suceder sólo cuando la operación entrante tiene éxito. La figura 3.9 muestra cómo debería ser utilizado el proxy cuando se envían consultas a éste.

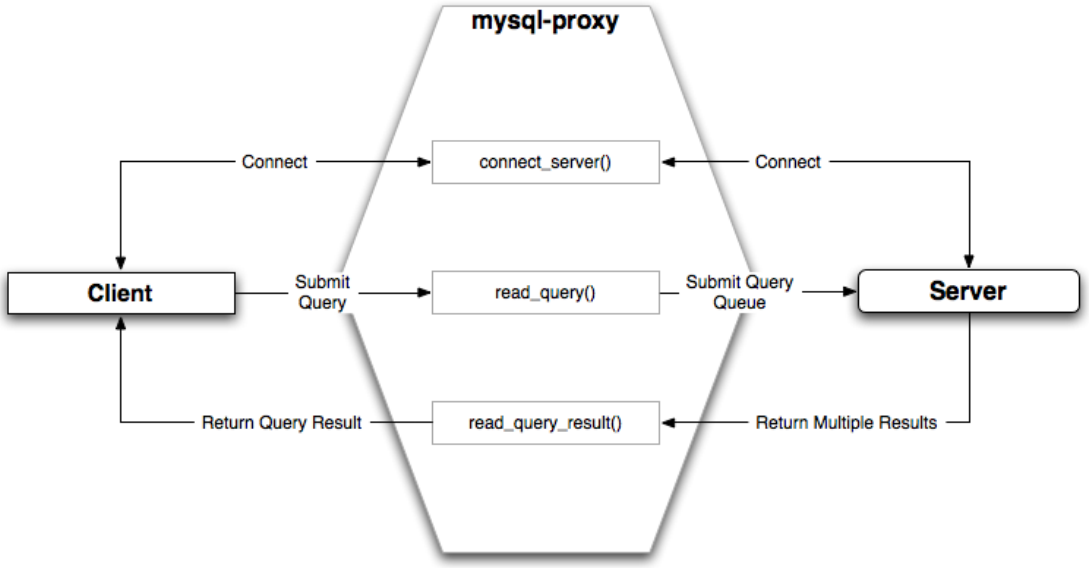

Figura 3.9: Arquitectura de MySQL-Proxy. Fuente: [JanK12].

Para diseñar el comportamiento de MySQL-Proxy fue necesario programar un script en lenguaje LUA, el cual fue llamado **mi-proxy.lua**[6]. Este script se compone de varias funciones, las que se pueden dividir en dos grupos: las funciones principales, que son aquellas que precisa MySQL-Proxy para configurar su funcionamiento, y las funciones secundarias, que fueron implementadas como apoyo a estas funciones principales.

La programación del proxy también incluyó el manejo de un pool de conexiones, es decir, se definió un número de conexiones mínimo y máximo para cada servidor de Base de Datos, tanto de lectura como de escritura. En caso de que alguno de éstos posea un número mayor de conexiones al permitido, se descarta este backend en el algoritmo de selección del servidor. En caso de estar descartados todos los servidores, se escoge el servidor *maestro*.

3.5.1. Ejecución de una consulta en el proxy

El proceso que ejecuta el proxy ante una consulta es el siguiente.

1. La conexión es capturada por la función principal **connect_server()**. Se escoge pre-

[6]El código fuente de este script se puede encontrar en el Apéndice C

liminarmente un servidor dependiendo del pool de conexiones. Si todas las conexiones están tomadas entonces se selecciona el servidor maestro.

2. Se lee el resultado de la autorización en el backend en la función principal llamada **read_auth_result()**. Si ésta fue exitosa entonces se continúa.

3. Se captura la consulta en la función principal **read_query(packet)**. En *packet* se encuentra en el primer byte el tipo de instrucción (QUIT, QUERY, etc.) y desde el segundo byte en adelante la consulta.

   a) En caso de contar con una instrucción QUIT al cliente se le envía una instrucción de éxito, pero al backend no se le envía la orden de quitar la conexión, pues con el pool de conexiones se manejarán todas estas instrucciones.

   b) Si se obtiene una instrucción QUERY entonces se procesa el texto de la consulta para identificar cuál es el tipo de operación SQL a ejecutar. Una vez que se identifica el tipo de consulta, en caso de ser una operación de actualización se utiliza una función secundaria que detecta las tablas involucradas; en cambio, si se trata de una operación de selección ocurre el algoritmo de selección de servidor, el cual es el siguiente.

      1) Se crea una conexión con el servidor principal y se consulta por el listado de servidores actualizados.
      2) En caso de que el resultado de la consulta sea vacío se selecciona por omisión al servidor principal.
      3) Caso contrario, se itera por el listado de servidores del sistema y en caso de encontrar un servidor que sea sólo de lectura, que no esté caído, que esté en el listado de servidores actualizados y que tenga conexiones disponibles, entonces se marca ese servidor como el servidor que responderá ante la consulta.

4. En caso de que aún no se haya seleccionado algún servidor entonces se marca al servidor *maestro* como el encargado de responder a la consulta.

5. Se verifica nuevamente el tipo de consulta que se enviará al servidor.

a) En caso de no ser una operación de actualización entonces se encola la consulta directamente en el servidor *maestro*.

b) Si es una operación de actualización, corresponde actualizar en la tabla de replicación sólamente si la operación tiene éxito. Para ésto, se encapsula en una transacción tanto la actualización que se quiere realizar como la modificación en la tabla de replicación. Se encolan estas instrucciones en el servidor *maestro*.

6. Se envía la consulta al servidor correspondiente.

7. Se captura el resultado de la consulta en la función principal **read_query_result(inj)**, se ignoran los resultados de las consultas que fueron inyectadas y se envían al cliente los resultados de sus consultas.

3.5.2. Funciones principales y secundarias

El funcionamiento de MySQL-Proxy se configura mediante un script, el cual debe ser programado en lenguaje LUA. Si se implementan ciertas funciones específicas se puede modelar de forma bastante precisa el comportamiento del sistema de distribución de consultas. Estas funciones son invocadas en distintas etapas a lo largo del proceso de consulta, lo cual significa que el implementarlas de forma adecuada ayuda a mantener el control sobre el proxy en prácticamente todo el ciclo. Para este trabajo estas funciones fueron denominadas como funciones principales.

Las funciones principales implementadas en este trabajo y su descripción respectiva son las siguientes.

connect_server() Método que captura cualquier conexión entrante que MySQL-Proxy ha aceptado. En esta función se define un servidor que responderá a la consulta entrante, el cual puede ser modificado por la función *read_query()*.

read_auth_result() Método que lee el resultado de la autentificación de MySQL-Proxy con el servidor de backend seleccionado.

read_query(packet) Método que intercepta la consulta enviada al proxy y realiza la lógica para seleccionar el servidor de backend adecuado dependiendo de la consulta. Es en este

lugar donde se realiza el *rw-splitting*, es decir, si se trata de una operación de sólo lectura se escoge dentro de los servidores esclavos aquel que puede responder adecuadamente a la consulta, más si se trata de una consulta de escritura ésta se redirige al maestro, se capturan las tablas involucradas en la actualización y se marcan como desactualizadas en la tabla de replicación.

read_query_result(inj) Método que captura la respuesta desde el servidor de backend y envía la respuesta al cliente.

disconnect_client() Método que intercepta una operación de desconexión desde un cliente. Resetea el backend para el cliente.

Las funciones **read_handshake()** y **read_auth()** también configuran el comportamiento de MySQL-Proxy, pero para este sistema no fueron consideradas.

Las funciones secundarias, programadas específicamente para este trabajo, son las siguientes.

rows(connection, sql_statement) Función que retorna cada registro de una consulta sql, dada una consulta y una conexión.

search(table, value) Función que retorna `true` en caso de que se encuentre *value* en la tabla *table*.

extract_db_truncate(default_db, tokens_query) Función que extrae desde una consulta de tipo TRUNCATE la tabla sobre la cual se está operando.

extract_db_create(default_db, tokens_query) Función que extrae desde una consulta de tipo CREATE las tablas sobre la cual se realiza la operación.

extract_db_droptable(default_db, tokens_query) Función que extrae desde una consulta de tipo DROP TABLE las tablas sobre la cual se realiza la operación.

extract_db_update(default_db, token_query) Función que extrae desde una consulta de tipo UPDATE las tablas sobre la cual se realiza la operación.

extract_db_delete(default_db, tokens_query) Función que extrae desde una consulta de tipo DELETE las tablas sobre la cual se realiza la operación.

extract_db_load(default_db, tokens_query) Función que extrae desde una consulta de tipo LOAD DATA las tablas sobre la cual se realiza la operación.

split(str, sep) Función que separa el string *str* por el separador *sep*.

convert_to_where(list) Función que convierte un listado de tablas del tipo "cli.cli_bdm, test_db.test_table" en una cláusula SQL de tipo WHERE, del estilo "WHERE (rt.database = ´cli´ AND rt.table = ´cli_bdm´) OR (rt.database = ´test_db´ AND rt.table = ´test_table´)" para apuntar a esas tablas en una posterior consulta a la tabla de replicación.

convert_to_insert(list) Función que convierte un listado de tablas del tipo "cli.cli_bdm" en un listado del tipo "´cli´, ´cli_bdm´" para una posterior consulta de inserción a la tabla de replicación.

3.5.3. Operaciones implementadas

Las operaciones que se pueden realizar en una Base de Datos son de cinco tipos: Sentencias DDL, sentencias DML, sentencias de control de transacciones, de control de sesión y de control de sistema [Oracle12]. Los tipos de operaciones que fueron incluidas en el sistema son las siguientes.

Sentencias DDL

Las sentencias DDL son utilizadas para definir o modificar la estructura de las tablas o Bases de Datos que componen el sistema. En el programa desarrollado se incluyó soporte a las siguientes sentencias DML:

- CREATE
- TRUNCATE
- DROP TABLE

Sentencias DML

Este tipo de instrucciones acceden y manipulan datos de objetos existentes en la Base de Datos. Se incluyó en el sistema desarrollado soporte al siguiente tipo de instrucciones:

- SELECT
- INSERT
- UPDATE
- LOAD DATA
- DELETE

El resto de los tipos de operaciones se escapan de los alcances definidos para este proyecto y se proponen como trabajo futuro la correcta integración de éstos.

Las sentencias DDL de tipo ALTER TABLE, RENAME TABLE, CALL, REPLACE no fueron incluidas en el sistema, pero basta con modificar la lógica de la aplicación y crear una función secundaria que extraiga las tablas que serán actualizadas para integrarlas. Cuando el proxy recibe una operación de cualquiera de estos tipos envía un mensaje indicando "operación no soportada en replicador".

La sentencia DML LOAD DATA y la operación DDL CREATE TABLE deben ser realizadas mediante línea de comandos y no dentro de la aplicación cliente *mysql-ib* utilizada en los terminales. Esta limitante tiene que ver con el funcionamiento interno de la aplicación MySQL-Proxy y para cambiar este comportamiento se debe realizar una modificación en el código propio del programa, lo cual está fuera de los alcances de esta memoria.

Para actualizar una tabla el proxy intercepta la consulta e inyecta en una transacción las modificaciones a la tabla de replicación. El siguiente es un ejemplo de una transacción ante una consulta de tipo CREATE.

```
START TRANSACTION;

SET AUTOCOMMIT=0;
-- Esta es la consulta enviada por el cliente
UPDATE TABLE cliente.cliente_bdm SET ID=0;
```

```
-- Esta es la consulta inyectada para actualizar
-- en tabla de replicación
UPDATE Replication.r_tables rt
SET status = -1, modified_time=NOW()
WHERE (rt.database = 'cliente' AND rt.name='cliente_bdm');

COMMIT;
```

Con este control mediante transacciones se logra la atomicidad de la operación completa y la tabla de replicación se actualiza si y sólo si la consulta enviada por el cliente tiene éxito.

Capítulo 4

Resultados y Conclusiones

Para cuantificar la mejora en el desempeño con el sistema de alta disponibilidad fueron realizaron mediciones de tiempo de ejecución, las que fueron contrastadas con el escenario sin el software implementado. En específico fueron calculados el tiempo de ejecución de una carga de datos y de respuesta ante consultas simultáneas.

En este capítulo se presentan tablas, gráficos e imágenes de los resultados obtenidos con las aplicaciones desarrolladas. En base a éstos se desarrollan las conclusiones sobre el presente trabajo de memoria.

4.1. Ambiente de pruebas

El ambiente de pruebas montado consta de cuatro máquinas virtuales, las cuales son alojadas en un computador con sistema operativo Ubuntu 10.04 LTS (*Lucid Lynx* con 4 GB de memoria RAM y un procesador Intel i5 M450 de 2.4 Ghz.

La máquina virtual **Ubu** posee como sistema operativo a Ubuntu Server versión 10.04.3 LTS de 32 bits, con memoria RAM instalada de 640 MB. **Ubu** tiene instalada la versión de Infobright IEE 4.0.5 con una licencia de evaluación comercial, la cual no tiene costo de uso asociado. Las otras máquinas virtuales, **Ubu1**, **Ubu2** y **Ubu3**, tienen instalado Ubuntu Server versión 10.04.3 LTS de 32 bits como sistema operativo, con memoria RAM de 640 MB. Estas últimas cuenta con la versión de Infobright ICE 4.0.6.

Cada una de las máquinas virtuales fue configurada para utilizar un procesador independiente de los cuatro disponibles del sistema que las aloja. Las pruebas realizadas fueron ejecutadas bajo una red de conexión inalámbrica a 1 Mb/s.

4.2. Tiempos sistema de replicación

Para realizar las pruebas de desempeño de la aplicación de replicación fueron utilizados distintos *dumps* de datos obtenidos desde las Bases de Datos de los clientes de Penta Analytics. El número total de registros de las tablas se escogieron de tal forma que todos fuesen distintos entre sí.

Se midió también el tiempo que toma a un operador de Base de Datos de la empresa el descargar, copiar y cargar los datos entre las máquinas **Ubu** y **Ubu1**. Este tiempo se utilizó como símil de un sistema de replicación "manual", el cual representa la situación actual.

Los tiempos que tardaron el sistema de replicación y el operador de Base de Datos en realizar las operaciones respectivas quedan reflejados en la tabla 4.1 y en la figura 4.1.

Cantidad de registros	**Tiempo operador (segundos)**	**Tiempo aplicación (segundos)**
806555	136	18
2677204	230	54
9590774	328	190
18750487	572	396
27130421	685	542

Cuadro 4.1: Tiempo de proceso de replicación por cantidad de registros

4.3. Tiempos sistema de distribución de consultas

La ejecución de las pruebas de desempeño para el sistema de distribución de consultas constó, en primer lugar, de una consulta SQL que involucra dos tablas, provocando un cruce de datos de 27130421 registros contra otros 7668. Esta consulta fue enviada simultáneamente, en grupos desde 1 hasta 5, hacia el servidor **Ubu** y hacia el sistema de distribución de consultas, con **Ubu** y **Ubu1** como servidores *maestro* y *esclavo*, respectivamente.

Lo que se busca medir con estas consultas simultáneas es el tiempo de respuesta obtenido para cada conjunto de consultas y así cuantificar en promedio el tiempo de respuesta de ambos escenarios.

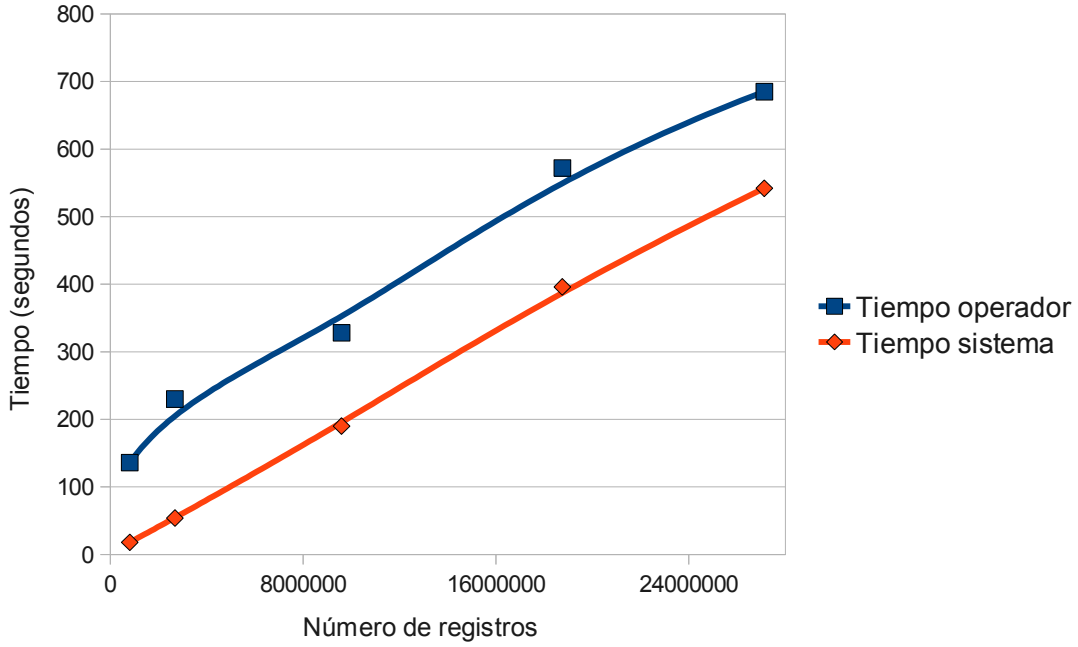

Figura 4.1: Gráfico de tiempo v/s cantidad de registros para replicación mediante operador y aplicación. Fuente: elaboración propia en base a resultados de la tabla 4.1

La ejecución de la batería de pruebas fue programada en un script *bash*, el cual envía las consultas a los sistemas respectivos en paralelo y cronometra mediante el comando *time* el tiempo que tarda cada consulta en ser respondida.

Los resultados obtenidos de esta prueba se pueden apreciar en la tabla 4.2.

Por cada conjunto de datos obtenido para cada ejecución de consultas simultáneas, se calculó el tiempo promedio de respuesta para ese sistema. Este resultado evidencia cuánto tiempo se estima que tarde en responder aproximadamente el sistema ante el escenario planteado.

El tiempo promedio de respuesta para cada sistema, en cada escenario, se puede apreciar en la figura 4.2.

Para evaluar el impacto de incluir más nodos en el sistema de distribución de carga, es decir más servidores de Bases de Datos, se realizaron mediciones de tiempo con otra consulta SQL que involucra dos tablas, provocando un cruce de datos de 3671917 registros contra otros 180. Esta consulta fue enviada simultáneamente, en grupos desde 1 hasta 20, hacia el sistema de distribución de consultas configurado con 1,2,3 y 4 nodos respectivamente.

El tiempo promedio de respuesta del sistema se muestra en la figura 4.4.

Consultas simultáneas	Tiempo consultas sin proxy (segundos)	Tiempo consultas con proxy (segundos)
1	23.11	33.45
2	43.89 47.07	34.07 26.38
3	66.21 66.15 65.65	26.78 66.87 66.92
4	101.63 101.74 101.63 101.63	72.11 72.14 50.26 50.02
5	136.81 225.71 225.71 225.78 137.75	84.45 84.44 72.25 72.28 85.34

Cuadro 4.2: Tiempo de respuesta de consultas por cantidad de consultas simultáneas

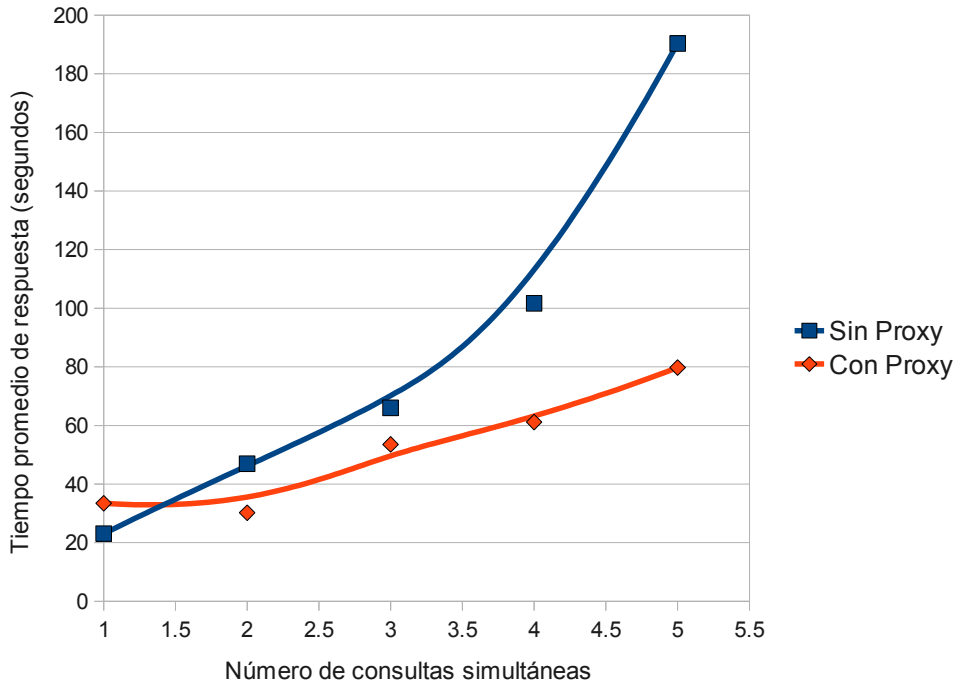

Figura 4.2: Gráfico de cantidad de consultas simultáneas v/s tiempo promedio de ejecución. Fuente: elaboración propia en base a resultados de la tabla 4.2.

		Nodos			
		1 Nodo	2 Nodos	3 Nodos	4 Nodos
Número de consultas concurrentes	1	2.72	2.70	2.64	2.62
	2	5.34	3.01	2.62	2.70
	3	7.96	4.77	2.86	2.88
	4	10.47	7.01	4.29	2.95
	5	13.20	7.59	5.33	3.99
	6	16.08	9.08	6.74	5.65
	7	18.47	10.36	7.89	7.66
	8	20.98	12.21	9.40	7.52
	9	23.74	14.31	10.84	8.28
	10	26.37	14.86	12.19	9.39
	11	29.29	16.18	13.86	10.73
	12	31.83	17.72	15.09	13.45
	13	34.15	19.00	15.71	13.27
	14	36.93	20.98	15.87	14.51
	15	39.41	22.30	17.18	16.54
	16	41.97	23.51	21.86	16.90
	17	45.26	24.92	21.87	17.62
	18	48.06	26.52	23.58	18.55
	19	50.56	28.87	23.51	22.15
	20	52.89	30.87	24.92	22.55

Figura 4.3: Tabla de tiempo de respuesta promedio en segundos ante consultas por cantidad de consultas simultáneas y nodos. Fuente: elaboración propia.

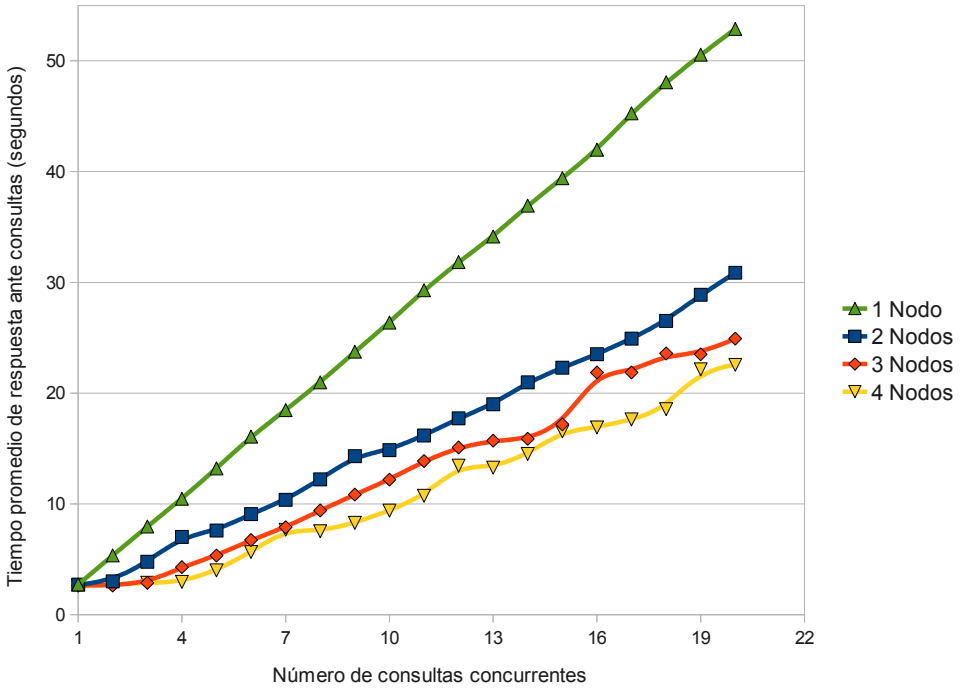

Figura 4.4: Gráfico de cantidad de consultas simultáneas v/s tiempo de ejecución para distinto número de nodos. Fuente: elaboración propia en base a resultados de la tabla 4.3.

4.4. Consistencia del sistema de alta disponibilidad

La primera verificación realizada en el sistema consistió en asegurar que los datos cargados en los servidores esclavos fuesen exactamente los mismos que los originales. El sistema de archivos distribuido NFS aseguró que los datos a repartir no fueran corruptos durante el transporte de éstos. Por otro lado, se comprobó que los datos tanto en cantidad como en integridad individual fuesen una réplica de los datos del servidor maestro; para ésto fueron contrastados con los datos de origen, obteniéndose óptimos resultados.

Para asegurar la consistencia del sistema se consideraron situaciones de riesgo en las cuales el sistema podría quedar con información errónea sobre el estado de actualización. Particularmente es de gran interés el verificar que el estado de actualización de cada tabla registrado en el sistema sea consistente con el estado real de actualización.

Los escenarios planteados fueron los siguientes.

- Pérdida de conexión con servidor principal.
- Pérdida de conexión con servidor esclavo.
- Corte de corriente.
- Base de Datos *maestra* o *esclava* no disponible.
- Falla en programa principal de replicación.
- Falla en programa periférico.

Se preparó cada escenario y se expuso el sistema a estos potenciales riesgos. La respuesta del sistema en cada caso se puede visualizar en la tabla 4.3.

4.5. Escalabilidad y Paralelismo

Se analizaron estas propiedades para el sistema desarrollado buscando identificar cómo mejorar el rendimiento actual del sistema. Para analizar la escalabilidad y el paralelismo del sistema total se analizará por separado cada una de las aplicaciones que la componen.

Evento	Descripción	Comportamiento del sistema
Caída proceso replicacion	El proceso de replicación se detiene en cualquiera de sus etapas	Se captura excepción y el sistema continúa su ejecución. Aún cuando se lanzó una transacción, el gestor de BD realizó la operación de *rollback* (deshacer cambios)
Caída programa de *dump*	El proceso de *dump* de alguna BD (Base de Datos) se detiene	El programa periférico retorna un valor indicando errores y el sistema no continúa actualizando esa tabla para ese servidor. El programa no alcanza a modificar estado de actualización
BD replicación no disponible	No se puede conectar a la BD de replicación	El programa lanza una excepción y el proceso de replicación se detiene. No se cambia el estado en la tabla
BD esclava no disponible	No se puede conectar a la BD *esclava*	El programa periférico retorna un valor indicando errores y el sistema no continúa actualizando esa tabla para ese servidor. Se mantiene estado de actualización sin altera
BD maestra no disponible	No se puede conectar a la BD *maestra*	El programa periférico retorna un valor indicando errores y el sistema no continúa actualizando esas tablas
BD no existente en BD *maestra*	No existe la BD a la cual se quiere consultar	El programa periférico retorna un valor indicando errores y el sistema no continúa actualizando esa tabla. No se modifica estado de actualización de las tablas
BD no existente en BD *esclava*	No existe la BD a la cual se quiere consultar	El programa periférico retorna un valor indicando errores y el sistema no continúa actualizando esa tabla para ese servidor
Tabla no existente en BD *maestra*	No existe la tabla que se quiere consultar	El programa periférico retorna un valor indicando errores y el sistema no continúa actualizando esa tabla. No se realiza cambio de estado de actualización de la tabla.
Pérdida de comunicación con servidor *maestro*	Por algún problema en la red (cable desenchufado p.ej.) se interrumpe la comunicación con el servidor *maestro*	Se captura la excepción rmi por cada error. No se actualizan tablas.
Pérdida de comunicación con servidor *esclavo*	Por algún problema en la red (cable desenchufado p.ej.) se interrumpe la comunicación con el servidor *esclavo*	Se captura la excepción rmi. No se marca tabla como actualizada.
No hay comunicación con servidor maestro	No se puede contactar al servidor maestro	Se captura la excepción y se finaliza el proceso, sin cambiar estado de las tablas
No hay comunicación con servidor esclavo	No se puede contactar al servidor esclavo	Se captura la excepción y no se actualiza esa tabla para ese servidor

Cuadro 4.3: Respuesta del sistema de distribución de carga ante diversos eventos.

4.5.1. Replicación

Esta aplicación actualmente no realiza su ejecución en paralelo. En particular cuando se efectúan las operaciones de actualización por cada Base de Datos desactualizada, la ejecución de la llamada a las aplicaciones periféricas de descarga y carga se realiza de forma secuencial.

Para que el sistema sea paralelizable se debe lanzar en un nuevo proceso cada una de estas llamadas. Incluso, eventualmente se podría designar a una máquina externa la labor de controlar la descarga o carga de datos por cada Base de Datos; es decir, contar con una máquina dedicada a la carga y descarga de una Base de Datos.

En cuanto a la escalabilidad, se analizó la escalabilidad horizontal y vertical. Para el primer caso, dado que el sistema no es actualmente paralelizable, el añadir más nodos al procesamiento no hará mejora alguna en el rendimiento, por ende actualmente el sistema no cuenta con este tipo de escalamiento. En cambio, si se mejoran las características del hardware de la máquina que ejecuta el proceso de replicación y las máquinas que realizan la carga o descarga de datos (escalabilidad vertical), ésto traerá consigo una mejora sustancial en el rendimiento del sistema.

4.5.2. Distribución de consultas

En este caso de las distribución de consultas, el añadir más nodos al sistema y configurar correctamente el programa MySQL-Proxy, según los resultados obtenidos los tiempos de respuesta mejoran sustancialmente.

Por otro lado, el mejorar el hardware de la máquina en donde el proxy es procesado junto las máquinas servidoras de datos también radica en una mejora en tiempo, pues es posible realizar más operaciones por unidad de tiempo. Los puntos anteriores permiten concluir que el proxy es escalable tanto horizontal como verticalmente.

Analizando el paralelismo, es posible configurar también el MySQL-Proxy haciendo que otras instancias del mismo programa, incluso no instaladas en la misma máquina, respondan ante una determinada petición. En este caso el proxy actúa como un verdadero balanceador de carga, redirigiendo las consultas a las otras instancias del proxy que puedan responder de forma adecuada ante la petición.

4.6. Conclusiones

El uso de un sistema de distribución de consultas y replicación automatizada de datos en un *clúster* de servidores de Bases de Datos, permite una optimización importantísima en el uso de recursos con respecto al escenario en que los servidores trabajan de forma aislada unos de otros.

El sistema de replicación, obtenido como resultado del desarrollo de este tema de memoria, evidencia una clara mejora en tiempos totales de ejecución de procesos de consultas, en comparación a la situación actual en donde los operadores de Bases de Datos descargan y mueven los datos de un servidor a otro para respaldar los datos.

Los tiempos obtenidos de los operadores de Bases de Datos representan una cota inferior de los tiempos reales que tardan éstos en trasladar la información. Durante la prueba éstos mantuvieron su foco completamente en la operación que estaban realizando, siendo que en su rutina cotidiana éstos efectúan también otras labores en paralelo. Incluso en ocasiones los operadores no notan que un proceso de *dump* o de copia de datos llegó a su fin; esta situación es más frecuente cuando se trata de operaciones que tardan un tiempo prolongado (por ejemplo más de dos horas). Aún considerando que los resultados recopilados representan el mínimo tiempo que pueden tardar, éstos no son suficientes como para equiparar los resultados que obtiene el sistema de replicación.

Se debe considerar además que el sistema, al ser automatizado, minimiza la probabilidad de errores que puede cometer el operador. Estos errores se pueden traducir en horas de trabajo desperdiciadas, al ser los *dumps* de datos que maneja la empresa de un volumen ostensiblemente grande para algunos clientes. Al liberar al operador de la necesidad de vigilar la operación cada determinado margen de tiempo, la empresa incurre en una mejora en la productividad de este empleado, lo cual repercute en un mejor desempeño económico y una mayor rentabilidad.

En una visión más general, el sistema respondió de forma adecuada a los eventos que hubiesen podido mermar su consistencia e integridad. Con los ejercicios realizados se comprobó que la solución obtenida es robusta, gracias a las operaciones atómicas implementadas en los procesos de cambio de estado. Se mostró además que ante cualquier corte en alguna

operación, por diversas eventualidades, el sistema completo se mantiene consistente y puede recuperar su último estado para su continuidad operativa inmediata.

Los tiempos de respuesta del sistema de distribución de consultas ante las pruebas realizadas mostraron que el desempeño de éste en tiempos promedios supera claramente a los tiempos de respuesta de sólo un servidor de Bases de Datos (ver figura 4.2). La tendencia mostrada en las figura 4.2 indica que ante mayor cantidad de consultas simultáneas en el sistema, mayor es la amplitud con que el proxy supera a la situación actual.

Los servidores de prueba poseen características similares en cuanto a recursos de hardware, por tanto sus tiempos de respuesta ante una consulta no debiesen variar demasiado entre sí. En el contexto del sistema analizado, la diferencia encontrada se explica por dos razones: El algoritmo de selección del servidor que debe atender la consulta y los recursos de hardware utilizados en él.

El algoritmo Round-Robin implementado tiene la característica, para el caso con dos servidores, que la elección del backend se realiza de forma intercalada entre cada servidor de Base de Datos. De esta forma se asegura que el servidor que atiende la primera consulta entrante no atiende a la segunda, a menos que haya ocurrido una operación de actualización de datos.

La diferencia en tiempo promedio de respuesta se hace notoria desde la cuarta consulta simultánea en adelante. En este caso, el tiempo de respuesta del servidor dedicado (sin proxy) guarda relación con la fragmentación de los recursos de éste para responder a las cuatro consultas, las cuales deben competir entre sí para aprovechar al máximo la fracción de tiempo que se les asignará. En cambio, el tiempo de respuesta del proxy depende del desempeño de dos servidores, cada uno de los cuales atiende dos consultas y los cuales deben fragmentar sus recursos en apenas dos tareas, respectivamente.

Los resultados obtenidos en un clúster de Bases de Datos con más de dos servidores muestran que el sistema responderá de mejor forma mientras mayor sea el número de servidores actualizados y disponibles para el proxy.

El impacto de la utilización del proxy y el script de procesamiento de consultas en los tiempos de respuesta de las Bases de Datos no fue significativo. El sobrecosto no superó los pocos segundos y en algunos casos la diferencia fue imperceptible (algunos milisegundos) y

se trata de un costo fijo por cada consulta a la Base de Datos.

Para mejorar el desempeño del sistema, en el caso del proxy basta con aumentar los nodos involucrados en la respuesta ante las consultas, mejorar el hardware de las máquinas involucradas en el proceso o configurar un pequeño *clúster* de aplicaciones MySQL-Proxy. En cuanto al sistema de replicación se requiere de una modificación en el código para añadir paralelismo al sistema y con ésto mejorar el desempeño. Pero si se mejora de forma independiende el hardware del equipo donde es ejecutado el proceso, el sistema responderá de mejor forma para el proceso de replicación.

La integración de este sistema en el ambiente de trabajo actual no requiere de grandes modificaciones adicionales a una instalación estándar de este sistema. Es decir, basta con instalar el lenguaje Java en cada uno de los componentes del *clúster*, instalar MySQL-Proxy en algún servidor, crear la tabla de Replicación con los datos necesarios de los servidores involucrados, configurar el sistema de archivos compartidos NFS y finalmente cambiar la conexión que antes apuntaba directamente a los servidores para que se conecte con MySQL-Proxy.

Capítulo 5

Trabajo Futuro

El trabajo desarrollado durante el semestre entregó como resultado un sistema de alta disponibilidad con estados consistentes en cada uno de sus procesos. A pesar del avance que éste representa, aún existen variadas tareas con las que se puede continuar mejorando la solución implementada.

5.1. Sistema de replicación

Para el sistema de replicación existen varios tópicos en los cuales se pueden continuar explorando mejoras.

En cuanto a la tabla de replicación se puede desarrollar una aplicación que provea una interfaz gráfica para la tarea de añadir o eliminar servidores del sistema. Actualmente esta labor se realiza manualmente, método que con el tiempo puede llegar a ser bastante engorroso. Una mejora conveniente consiste en un programa que se conecte con la Base de Datos principal para leer sus tablas y así evitar el introducir una a una las tablas que se desean replicar.

Otro avance importante consiste en un método que provea alertas, vía correo electrónico u otra forma de mensajería, sobre los cambios de estado de las Base de Datos pertenecientes al sistema. A pesar de que el sistema de distribución de carga tiene la labor de discernir entre los servidores actualizados o desactualizados, actualmente no existe alguna forma para detectar automáticamente cuando un servidor completo está desactualizado o cuando es sólo el maestro quien está actualizado.

La forma en que el sistema de replicación actualmente realiza sus iteraciones (primero por Bases de Datos, después por tablas y finalmente por servidores) busca minimizar los efectos

en el proceso completo relacionados con la falla en alguna de estas repeticiones. En caso de que el sistema falle en la iteración por servidores, ésta afecta sólo a esa tabla de ese servidor, sin afectar la copia de esa tabla en el resto de los servidores.

En caso de cambiar la manera de iterar, por ejemplo comenzando por los servidores, siguiendo por las Bases de Datos y terminando con las tablas, una falla en la conexión con el cliente rmi hará que se detenga la copia de datos a ese servidor y se desactualice casi por completo; más, en un caso optimista, se pueden ejecutar optimizaciones de los *dumps* de datos, descargando más de una tabla a la vez. Estos cambios implican una modificación importante en el código y la lógica del sistema, y se debe realizar un análisis comparativo contrastando con la solución implementada y considerando la probabilidad de fallas del sistema.

5.2. Sistema de distribución de carga

Una mejora evidente para este sistema consiste en explorar otros tipos de algoritmos de selección de servidor. El algoritmo actualmente utilizado selecciona un servidor *esclavo* si y sólo si todas las tablas que componen a esa Base de Datos están actualizadas. Esto trae consigo que en ocasiones un servidor sea descartado aún cuando las tablas que efectivamente están involucradas en la consulta se encuentren actualizadas.

La limitante anterior se debe al proceso de identificación de las tablas involucradas en una consulta de selección. Acá una mejora sustancial para el sistema radica en programar un método en el script LUA del proxy que, dada una consulta SQL de selección, entregue las tablas que serán utilizadas.

Otra mejora en el algoritmo de selección de servidor trata sobre la obtención de la carga de los servidores. Esta característica debe ser sujeta a una minuciosa evaluación, pues el tiempo involucrado en la recolección de esta carga por cada servidor puede hacer que la consulta tome un tiempo mayor a la mejora detectada para el sistema, lo cual para cierto número de consulta concurrentes puede resultar menos eficiente que la situación actual con el trabajo del proxy.

5.3. Palabras finales

Como se observa, este desarrollo abre varios caminos para investigar y desarrollar en base a este primer esfuerzo. La solución conseguida representa para Penta Analytics un ejemplo de inversión costo-efectiva de recursos, marcando una innovación en el mercado, y los temas propuestos como trabajo futuro muestran un abanico interesante de alternativas para continuar estudiando los temas de replicación y distribución de carga expuestos en este tema de memoria.

Referencias

[BSDO04] Balasubramanian, Jaiganesh, Douglas C. Schmidt, Lawrence Dowdy, y Ossama Othman. «Evaluating the Performance of Middleware Load Balancing Strategies.» En *Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International*. IEEE Computer Society, Washington, DC, USA, 2004, 135–146.

[BeHG87] Bernstein, P.A., V. Hadzilacos, y N. Goodman. *Concurrency control and recovery in database systems*. Addison-Wesley series in computer science. Addison-Wesley Pub. Co., 1987.

[CKKS02] Cody, W. F., J. T. Kreulen, V. Krishna, y W. S. Spangler. «The integration of business intelligence and knowledge management.» *IBM Syst. J.*, 41, nº 4, (2002), 697–713. URL `http://dx.doi.org/10.1147/sj.414.0697`.

[CoGearm11] Comunidad de programadores de Gearman. «Esquema de funcionamiento del framework Gearman.» Última visita en Julio 2011, URL `http://gearman.org/images/gearman_stack.png`.

[CoGear11] Comunidad de programadores de Gearman. «Gearman.» Última visita en Julio 2011, URL `http://gearman.org/index.php`.

[CoSha11] Comunidad de usuarios de Shard-Query. «Esquema del funcionamiento de Shard-Query ante la ejecución de una consulta SQL.» Última visita en Julio 2011, URL `http://shard-query.googlecode.com/files/shard_query_dataflow.jpeg`.

[Cont11] Continuent Inc. «Arquitectura del sistema Tungsten.» Última visita en Julio 2011, URL https://s3.amazonaws.com/releases.continuent.com/doc/replicator-2.0/html/Tungsten-Replicator-Guide/replicatorguide-commercial/figure/replicationarchitecture_html.png.

[Conti11] Continuent Inc. «Tungsten Replicator.» Última visita en Febrero 2012, URL http://code.google.com/p/tungsten-replicator/.

[DaPV08] Dasgupta, S., C.H. Papadimitriou, y U.V. Vazirani. *Algorithms*. McGraw-Hill Higher Education, 2008.

[French11] French, Steven. «A New Network File System is Born: Comparison of SMB2, CIFS and NFS.» En *Proceedings of the Ottawa Linux Symposium*. Ottawa Linux Symposium, 2007.

[Indi11] Indiana University. «University Information Technology Services, Knowledge Base: What is SQL?», Julio 2011. URL http://kb.iu.edu/data/ahux.html.

[Info08] Infobright Inc. «Infobright Community Edition, Technology White Paper.» *Inf. téc.*, 47 Colborne Lane, Suite 403, Toronto, Ontario M5E 1P8, Canada, Septiembre 2008.

[Info10] Infobright Inc. *Migration Guide: MySQL/MyISAM to Infobright*, Marzo 2010.

[IEEICE11] Infobright Inc. «Top 10 Differences: Infobright Enterprise Edition and Infobright Community Edition.» *Inf. téc.*, Infobright Inc., 2011. URL http://www.infobright.org/downloads/doc/IEE_vs_ICE4.0.pdf.

[IOWA11] Iowa State University, Ames Laboratory, U.S. Departament of Energy. «What is a Cluster Computer?» Última visita en Julio 2011, URL http://www.scl.ameslab.gov/Projects/parallel_computing/cluster_basics.html.

[JanK12] Jan Kneschke. «Arquitectura del sistema MySQL-Proxy.» Última visita en Febrero 2012, URL http://dev.mysql.com/doc/refman/5.1/en/images/proxy-architecture.png.

[LLSG92] Ladin, Rivka, Barbara Liskov, Liuba Shrira, y Sanjay Ghemawat. «Providing high availability using lazy replication.» *ACM Trans. Comput. Syst.*, 10, (1992), 360–391.

[MaSt03] Marcus, E., y H. Stern. *Blueprints for High Availability*. Wiley Pub., 2003.

[May10] May, Mar Oo. «Fault Tolerance by Replication of Distributed Database in P2P System Using Agent Approach.» *International Journal of Computers*, 4.

[MPGP01] Moiz, Salman Abdul, Sailaja P., Venkataswamy G., y Supriya N. Pal. «Article: Database Replication: A Survey of Open Source and Commercial Tools.» *International Journal of Computer Applications*, 13, nº 6, (2011), 1–8. Publicado por Foundation of Computer Science.

[Oracl11] Oracle Corporation. «Oracle Corporation; MySQL 5.0 Reference Manual: 5.2. MySQL Server Logs.», Julio 2011. URL http://dev.mysql.com/doc/refman/5.0/en/server-logs.html.

[Orac11] Oracle Corporation. «Oracle9i Database Concepts, Release 2 (9.2): 17. Triggers.», Julio 2011. URL http://download.oracle.com/docs/cd/B10500_01/server.920/a96524/c18trigs.htm.

[Oracle12] Oracle Corporation. «Types of SQL Statements.», Abril 2012. URL http://http://docs.oracle.com/cd/B14117_01/server.101/b10759/statements_1001.htm.

[Paul08] Paul, Sujoy. *Pro SQL Server 2008 Replication*. Apress, Berkely, CA, USA, 2009, 2da edón.

[Pent11] Penta Analytics S.A. Última visita en Febrero 2012, URL http://www.analytics.cl.

[RedH11] Red Hat Inc. «Red Hat Enterprise Linux 3: System Administration Guide: Chapter 38. Log Files.», Julio 2011. URL `http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/3/html/System_Administration_Guide/ch-logfiles.html`.

[RBSc00] Röhm, Uwe, Klemens Böhm, y Hans-Jörg Schek. «Olap Query Routing and Physical Design in a Database Cluster.» En *EDBT '00 Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology* (Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl, y Torsten Grust, eds.). Springer-Verlag, 2000.

[TaGi08] Tantisiriroj, W., y G. Gibson. «Network File System (NFS) in High Performance Network.», 2008. URL `http://institute.lanl.gov/isti/irhpit/projects/nfshpn.pdf`.

[XYEW98] Xu, Jian, Yinyan Cao, Ee-Peng Lim, y Wee-Keong Ng. «Database Selection Techniques for Routing Bibliographic Queries.» En *Proceedings of the 3rd ACM International Conference on Digital Libraries (DL'98)*. 1998.

[ZuPe08] Zuikevičiūtė, Vaidė, y Fernando Pedone. «Conflict-aware load-balancing techniques for database replication.» En *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, New York, NY, USA, 2008, SAC '08, 2169–2173.

Apéndices

A . Package `cl.analytics.replication`

A .1. Código clase `Replicator.java`

```
1  package cl.analytics.replication;
2
3  import org.apache.log4j.*;
4  import cl.analytics.utils.*;
5
6  import java.util.List;
7  import java.text.SimpleDateFormat;
8  import java.util.Calendar;
9
10 /**
11  * @author Ronald Poillot
12  */
13 public class Replicator {
14
15     static Config cfg;
16     static Logger logger;
17     static DBOperations dbo;
18
19     /**
20      * Metodo principal de replicacion.
21      * @param args
22      */
23     public static void main(String[] args) {
24         logger = Logger.getLogger(Replicator.class);
25         SimpleDateFormat formatter = new SimpleDateFormat("yyyy/MMM/dd HH:
               mm:ss");
26         Calendar initDate, endDate;
27
28         while (true) {
29             try {
30                 /**
31                  * Lectura de archivo de configuracion
32                  * Debe estar en la raiz del proyecto
33                  */
34                 initDate = Calendar.getInstance();
35                 logger.info("Inicio proceso de replicacion con fecha "
36                         + formatter.format(initDate.getTime()));
```

```
37                  logger.info("Obteniendo datos de archivo de propiedades");
38                  Config config = new Config("replication.properties");
39
40                  /**
41                   * Se crea instancia de conexion a BD
42                   * maestra
43                   */
44                  dbo = new DBOperations(
45                          config.getString("dbhost"),
46                          config.getString("dbname"),
47                          config.getString("dbuser"),
48                          config.getString("dbpass"),
49                          "infobright");
50
51                  /**
52                   * Se obtienen todas las BD desde
53                   * BD de replicacion
54                   */
55                  logger.info("Obteniendo bases de datos...");
56                  List<String> databases = dbo.queryDatabases();
57
58                  /**
59                   * Se itera por las bases de datos
60                   */
61                  for (String database : databases) {
62
63                      /*
64                       * Se obtienen las tablas de esas bases de datos
65                       */
66                      logger.info("Obteniendo tablas...");
67                      List<String> tables = dbo.queryTables(database);
68
69                      for (String table : tables) {
70
71                          /*
72                           * Por cada tabla se obtiene el listado de
                                servidores
73                           * desactualizados
74                           */
75                          logger.info("Obteniendo servidores desactualizados
                                ...");
76                          List<String> servers = dbo.queryServers(database,
                                table);
77
78                          /*
79                           * Si el listado de tablas obtenido es vacio
80                           * entonces no se debe replicar
81                           */
82                          if (servers.isEmpty()) {
83                              logger.info("Nada que replicar para base de
                                    datos " + database + " y tabla " + table);
84                          } else {
85                              int dump_result = 1;
86
```

```
87                                  /*
88                                   * Logica previa. Marca a los servidores con
                                        -2 si es que no hay
89                                   * servidores actualizandose
90                                   *
91                                   */
92                                  logger.info("Ejecutando logica previa");
93                                  dbo.executePreviousLogic(database, table);
94
95                                  /*
96                                   * Se revisa si el dump existente esta
                                        actualizado
97                                   */
98                                  logger.info("Revisando si hay un dump
                                       actualizado");
99
100                                 List<String> serversUpdating = dbo.
                                       queryServersUpdating(database, table);
101
102                                 if (!serversUpdating.isEmpty()) {
103                                     logger.info("Hay un dump actualizado de BD
                                            " + database + " y tabla " + table + "
                                           .\nNo gatillar nuevo dump");
104                                     dump_result = 0;
105
106                                 } else {
107
108                                     logger.info("Ejecutando proceso de dump");
109                                     dump_result = ReplicationClient.
                                           executeProcess("dump",
110                                                 config.getString("dbhost"),
111                                                 database,
112                                                 table);
113                                 }
114
115                                 /*
116                                  * Si no fallo el proceso de descarga de datos
117                                  * se ejecuta proceso de carga en servidores
                                        esclavos
118                                  */
119                                 if (dump_result == 0) {
120                                     logger.info("Dump listo. Marcar servidores
                                            "
121                                             + "como 'actualizando'");
122
123                                     for (String server : servers) {
124                                         dbo.updateTableUpdating(server,
                                               database, table);
125                                     }
126
127                                     logger.info("Proseguir con carga de datos"
                                           );
128
129                                     /*
```

```
130                                 * Por cada servidor se ejecuta la carga
                                        de datos
131                                 */
132                                for (String server : servers) {
133                                    int load_result = 1;
134
135                                    load_result = ReplicationClient.
                                           executeProcess("load",
136                                            server,
137                                            database,
138                                            table);
139
140                                    if (load_result == 0) {
141                                        /*
142                                         * Si no fallo la carga entonces
143                                         * se marca a la tabla como
                                                actualizada
144                                         */
145                                        logger.info("Proceso de carga
                                               exitoso.\n"
146                                                + "Marcando tabla " +
                                                    database + "." + table
                                                    + " de " + server + "
                                                    como actualizada");
147                                        dbo.updateTableUpdated(server,
                                               database, table);
148                                    } else {
149                                        /*
150                                         * Fallo el load.
151                                         */
152                                        logger.info("Error en proceso de
                                               carga en esclavo " + server);
153                                    }
154                                }
155                            } else {
156                                /*
157                                 * Fallo el dump. Marcar como
                                        desactualizada again
158                                 */
159                                logger.info("Error en proceso de dump.");
160                            }
161                        }
162                    }
163                }
164
165            } catch (Exception e) {
166                logger.error("Error en proceso de Replicacion. Detalles: "
                       + e);
167            } finally {
168                endDate = Calendar.getInstance();
169                logger.info("Fin de proceso de replicacion con fecha "
170                        + formatter.format(endDate.getTime()));
171            }
172            try {
```

```
                    Thread.sleep(10 * 1000);
                } catch (Exception e) {
                    logger.error("Error al suspender proceso. Detalles: " + e);
                }
            }
        }
    }
```

A .2. Código clase DBOperations.java

```
package cl.analytics.replication;

import cl.analytics.utils.*;
import cl.analytics.utils.Connection;

import org.apache.log4j.Logger;
import java.sql.*;
import java.util.List;
import java.util.ArrayList;

/**
 * UpdateDBOperations es la clase que realiza operaciones sobre la BD del
 * sistema de replicacion
 *
 * @author Ronald Poillot
 */
public class DBOperations {

    private Logger logger = Logger.getLogger(DBOperations.class);
    private Operation op = null;

    /**
     * Constructor de la clase para conexiones MySQL
     *
     * @param dbhost URL de la Base de Datos
     * @param dbname Nombre de la Base de Datos
     * @param dbuser Username de la Base de Datos
     * @param dbpass Password de la Base de Datos
     */
    public DBOperations(String dbhost, String dbname, String dbuser,
            String dbpass, String type) {
        Connection conn = null;

        if (type.compareTo("mysql") == 0) {
            conn = new MySQLConnection(dbhost, dbname, dbuser, dbpass);
        } else if (type.compareTo("infobright") == 0) {
            conn = new InfobrightConnection(dbhost, dbname, 5029, dbuser,
                dbpass);
        }

```

```
40          this.op = new Operation(conn);
41      }
42
43      /**
44       * Metodo que entrega el listado de servidores desactualizados dada
             una tabla
45       *
46       * @param database la base de datos de la tabla
47       * @param table el nombre de la tabla
48       * @return lista de ip's de servidores desactualizados
49       */
50      public List<String> queryServers(String database, String table) {
51          String queryFormat = "SELECT DISTINCT rt.server "
52                  + "FROM Replication.r_tables rt "
53                  + "WHERE rt.database = '%s' "
54                  + "AND rt.name = '%s' "
55                  + "AND rt.status < 1;";
56
57          String query = String.format(queryFormat, database, table);
58          ResultSet rs = op.executeQuery(query, logger);
59          List<String> allServers = new ArrayList<String>();
60
61          try {
62              while (rs.next()) {
63                  allServers.add(rs.getString("server"));
64              }
65          } catch (SQLException ex) {
66              logger.error("Error al leer la informacion del resultset", ex)
                 ;
67          } finally {
68              op.close();
69          }
70          return allServers;
71      }
72
73      /**
74       * Metodo que entrega un listado de servidores con estado 0 (
             actualizando)
75       * dada una tabla
76       *
77       * @param database la base de datos de la tabla
78       * @param table el nombre de la tabla
79       * @return listado de ip's de servidores con estado 0 (actualizando)
80       */
81      public List<String> queryServersUpdating(String database, String table
         ) {
82          String queryFormat = "SELECT DISTINCT rt.server "
83                  + "FROM Replication.r_tables rt "
84                  + "WHERE rt.database = '%s' "
85                  + "AND rt.name = '%s' "
86                  + "AND rt.status = 0;";
87
88          String query = String.format(queryFormat, database, table);
89          ResultSet rs = op.executeQuery(query, logger);
```

```
90          List<String> allServers = new ArrayList<String>();
91
92          try {
93              while (rs.next()) {
94                  allServers.add(rs.getString("server"));
95              }
96          } catch (SQLException ex) {
97              logger.error("Error al leer la informacion del resultset", ex)
                    ;
98          } finally {
99              op.close();
100         }
101         return allServers;
102     }
103
104     /**
105      * Metodo que entrega listado de bases de datos del sistema de
             replicacion
106      *
107      * @return listado de nombres de bases de datos
108      */
109     public List<String> queryDatabases() {
110         String query = "SELECT DISTINCT rt.database "
111                 + "FROM Replication.r_tables rt ;";
112
113         ResultSet rs = op.executeQuery(query, logger);
114         List<String> allDatabases = new ArrayList<String>();
115
116         try {
117             while (rs.next()) {
118                 allDatabases.add(rs.getString("database"));
119             }
120         } catch (SQLException ex) {
121             logger.error("Error al leer la informacion del resultset", ex)
                    ;
122         } finally {
123             op.close();
124         }
125         return allDatabases;
126     }
127
128     /**
129      * Metodo que entrega listado de nombres de tablas del sistema de
             replicacion
130      * dada una base de datos
131      *
132      * @param database la base de datos a consultar
133      * @return listado de tablas de esa base de datos en el sistema
134      */
135     public List<String> queryTables(String database) {
136         String queryFormat = "SELECT DISTINCT rt.name "
137                 + "FROM Replication.r_tables rt "
138                 + "WHERE rt.database = '%s';";
139
```

```
140             String query = String.format(queryFormat, database);
141             ResultSet rs = op.executeQuery(query, logger);
142             List<String> allTables = new ArrayList<String>();
143 
144             try {
145                 while (rs.next()) {
146                     allTables.add(rs.getString("name"));
147                 }
148             } catch (SQLException ex) {
149                 logger.error("Error al leer la informacion del resultset", ex)
                        ;
150             } finally {
151                 op.close();
152             }
153             return allTables;
154         }
155 
156         /**
157          * Metodo que actualiza estado de una tabla a 1 (actualizada) en
                sistema
158          * de replicacion. Realiza LOCK sobre la tabla r_tables y utiliza
                transacciones.
159          *
160          * @param server el servidor a marcar como actualizado
161          * @param database la base de datos de la tabla a marcar como
                actualizada
162          * @param table el nombre de la tabla a marcar como actualizada
163          */
164         public void updateTableUpdated(String server, String database, String
               table) {
165             String queryFormat = "START TRANSACTION;"
166                     + "LOCK TABLE Replication.r_tables as rt WRITE;"
167                     + "SET @old_status = (SELECT rt.status "
168                     + "FROM Replication.r_tables as rt "
169                     + "WHERE rt.database='%s' "
170                     + "AND rt.name='%s' "
171                     + "AND rt.server='%s' );"
172                     + "UPDATE Replication.r_tables as rt "
173                     + "SET rt.status = (SELECT CASE WHEN @old_status = 0 THEN
                           1 ELSE @old_status END), modified_time = NOW() "
174                     + "WHERE rt.database='%s' "
175                     + "AND rt.name='%s' "
176                     + "AND rt.server='%s';"
177                     + "UNLOCK TABLE;COMMIT;";
178 
179             String pre_query = String.format(queryFormat, database, table,
                    server,
180                     database, table, server);
181 
182             String[] query = pre_query.split(";");
183 
184             op.executeBatch(query, logger);
185             op.close();
186         }
```

```
187
188         /**
189          * Metodo que actualiza estado de tablas a 0 (actualizando). Realiza
                LOCK sobre
190          * tabla r_tables y ocupa transacciones. En flujo normal cambia estado
                 de -2 a 0,
191          * si no encuentra estado -2 entonces deja el estado antiguo.
192          *
193          * @param server el servidor a marcar como actualizando
194          * @param database la base de datos de la tabla a marcar como
                actualizando
195          * @param table el nombre de la tabla a marcar como actualizando
196          */
197         public void updateTableUpdating(String server, String database, String
                table) {
198             String queryFormat = "START TRANSACTION;"
199                     + "LOCK TABLE Replication.r_tables as rt WRITE;"
200                     + "SET @old_status = (SELECT rt.status "
201                     + "FROM Replication.r_tables as rt "
202                     + "WHERE rt.database='%s' "
203                     + "AND rt.name='%s' "
204                     + "AND rt.server='%s' );"
205                     + "UPDATE Replication.r_tables as rt "
206                     + "SET rt.status = (SELECT CASE WHEN @old_status = -2 THEN
                            0 ELSE @old_status END), modified_time = NOW() "
207                     + "WHERE rt.database='%s' "
208                     + "AND rt.name='%s' "
209                     + "AND rt.server='%s';"
210                     + "UNLOCK TABLE;COMMIT;";
211
212             String pre_query = String.format(queryFormat, database, table,
                    server,
213                     database, table, server);
214
215             String[] query = pre_query.split(";");
216
217             op.executeBatch(query, logger);
218             op.close();
219         }
220
221         /**
222          * Metodo que marca a una tabla como desactualizada. Utiliza LOCK y
                transacciones.
223          * Cambia estado de cualquiera a -1, excepto si es 1.
224          *
225          * @param server el servidor a marcar como desactualizada
226          * @param database la base de datos de la tabla a marcar como
                desactualizada
227          * @param table la tabla a marcar como desactualizada
228          */
229         public void updateTableOutdated(String server, String database, String
                table) {
230             String queryFormat = "START TRANSACTION;"
231                     + "LOCK TABLE Replication.r_tables as rt WRITE;"
```

```
232                 + "UPDATE Replication.r_tables as rt "
233                 + "SET rt.status = -1, modified_time = NOW() "
234                 + "WHERE rt.database='%s' "
235                 + "AND rt.name='%s' "
236                 + "AND rt.server='%s' "
237                 + "AND rt.status <> 1;"
238                 + "UNLOCK TABLE;COMMIT;";
239
240         String pre_query = String.format(queryFormat, database, table,
                server,
241                 database, table, server);
242
243         String[] query = pre_query.split(";");
244
245         op.executeBatch(query, logger);
246         op.close();
247     }
248
249     /**
250      * Metodo que ejecuta una logica previa a la actualizacion.
251      * Dada una base de datos y una tabla, si existe algun servidor con
             estado 0
252      * entonces marca a todos los servidores con estado 0 excepto los que
             tienen
253      * estado 1. Caso contrario deja a los servidores con estado -2 salvo
             los que
254      * tienen estado 1.
255      *
256      * @param database la base de datos de la tabla sobre la cual ejecutar
              logica
257      * @param table la tabla sobre la cual ejecutar la logica
258      */
259     public void executePreviousLogic(String database, String table) {
260         String queryFormat = "START TRANSACTION;"
261                 + "LOCK TABLE Replication.r_tables as rt WRITE;"
262                 + "SET @old_status = (IF (EXISTS (SELECT count(*) "
263                 + "FROM Replication.r_tables rt "
264                 + "WHERE rt.database = '%s' "
265                 + "AND rt.name = '%s' "
266                 + "AND rt.status = 0 "
267                 + "GROUP BY rt.database, rt.name), 1, 0));"
268                 + "UPDATE Replication.r_tables as rt "
269                 + "SET rt.status = (SELECT CASE WHEN @old_status = 1 THEN
                        0 ELSE -2 END), modified_time = NOW() "
270                 + "WHERE rt.database='%s' "
271                 + "AND rt.name='%s' "
272                 + "AND rt.status <> 1;"
273                 + "UNLOCK TABLE;COMMIT;";
274
275         String pre_query = String.format(queryFormat, database, table,
                database, table);
276
277         String[] query = pre_query.split(";");
278
```

```
279             op.executeBatch(query, logger);
280             op.close();
281 
282         }
283     }
```

A .3. Código clase ReplicationClient.java

```
1   /*
2    * To change this template, choose Tools | Templates
3    * and open the template in the editor.
4    */
5   package cl.analytics.replication;
6 
7   import java.rmi.registry.LocateRegistry;
8   import java.rmi.registry.Registry;
9   import cl.analytics.replication.server.Commands;
10 
11  import org.apache.log4j.Logger;
12 
13  /**
14   *
15   * @author ronald
16   */
17  public class ReplicationClient {
18 
19      public static final Logger logger = Logger.getLogger(ReplicationClient
            .class);
20 
21      public ReplicationClient() {
22      }
23 
24      /**
25       * Metodo que hace llamado mediante rmi a servidores para hacer
              descarga o
26       * carga de datos, segun corresponda
27       *
28       * @param type la operacion a ejecutar, actualmente se soporta "dump"
              y "load"
29       * @param server el servidor al cual llamar, como ip
30       * @param database el nombre de la base de datos sobre la cual operar
31       * @param table tabla con la cual operar
32       * @return 1 en caso de exito, 0 en caso de error.
33       */
34      public static int executeProcess(String type,
35              String server,
36              String database,
37              String table) {
38 
39          int retVal = 1;
40 
```

```
41          if (System.getSecurityManager() == null) {
42              System.setSecurityManager(new SecurityManager());
43              logger.debug("Creando security manager");
44          }
45
46          try {
47              String name = "Commands";
48
49              logger.debug("Tratando de hacer bind con " + name);
50              logger.debug("La ip es " + server);
51
52              Registry registry = LocateRegistry.getRegistry(server);
53
54              logger.debug("Obtenido el registry");
55
56              Object obj = registry.lookup(name);
57              Commands cmds = (Commands) obj;
58
59              logger.debug("Conexion exitosa con servidor");
60
61              /*
62               * Se discrimina segun la variable type que accion ejecutar
63               */
64              if (type.equals("dump")) {
65                  retVal = cmds.executeDumpCompressProcess(database, table);
66                  logger.debug("Resultado de proceso de dump: " + retVal);
67              } else if (type.equals("load")) {
68                  retVal = cmds.executeDecompressLoadProcess(database, table
                        );
69                  logger.debug("Resultado de proceso de load: " + retVal);
70              } else {
71                  logger.error("Instruccion " + type + " no reconocida.
                        Abortando...");
72              }
73              logger.info("Proceso de " + type + " finalizado");
74
75          } catch (Exception e) {
76              logger.error("Error al conectar con rmi. Detalles: " + e);
77          } finally {
78              return retVal;
79          }
80      }
81  }
```

B . Package `cl.analytics.replication.server`

B .1. Código clase `Commands.java`

```
1   /*
2    * To change this template, choose Tools | Templates
```

```
 * and open the template in the editor.
 */
package cl.analytics.replication.server;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 *
 * @author ronald
 */
public interface Commands extends Remote {
    int executeDumpCompressProcess(String database, String table) throws
       RemoteException;
    int executeDecompressLoadProcess(String database, String table) throws
        RemoteException;
}
```

B .2. Código clase `CommandsImpl.java`

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package cl.analytics.replication.server;

import cl.analytics.utils.UsefulFunctions;

import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

import org.apache.log4j.Logger;

/**
 *
 * @author ronald
 */
public class CommandsImpl implements Commands {

    private static final Logger logger = Logger.getLogger(CommandsImpl.
       class);

    public CommandsImpl() {
        super();
    }

    public static void main(String[] args) {
        try {
            LocateRegistry.createRegistry(1099);
```

```
31              logger.info("RMI registry creado.");
32          } catch (Exception e) {
33              logger.error("Error iniciando RMI registry. Detalles:" + e);
34          }
35
36          if (System.getSecurityManager() == null) {
37              System.setSecurityManager(new SecurityManager());
38          }
39
40          try {
41              String name = "Commands";
42              Commands cmd = new CommandsImpl();
43              Commands stub = (Commands) UnicastRemoteObject.exportObject(
                    cmd, 0);
44
45              Registry registry = LocateRegistry.getRegistry();
46              registry.rebind(name, stub);
47              logger.info(name + " registrado correctamente.");
48          } catch (Exception e) {
49              logger.error("Excepcion en CommandsImpl. Detalles: " + e);
50          }
51      }
52
53      /**
54       * Metodo que ejecuta proceso de descarga, compresion de datos y
55       * copiado a directorio en red compartido
56       *
57       * @param database la base de datos a utilizar como String
58       * @param tables listado de tablas a descargar como lista de objetos
             Table
59       * @return 0 en caso de exito, >0 en caso de falla
60       * @throws RemoteException
61       */
62      @Override
63      public int executeDumpCompressProcess(String database, String table)
            throws RemoteException {
64          int retVal1 = 1;
65          int retVal2 = 1;
66
67          logger.info("Ejecutando descarga de datos");
68
69          logger.info("Recibida info. para procesar descarga desde base de
                datos :" + database);
70
71          /*
72           * Se crea en un arreglo los comandos a ejecutar
73           * para la descarga de datos
74           */
75          String[] command = new String[3];
76          command[0] = "scripts/dump_ib_db.sh";
77
78          try {
79              logger.info("Procesando tabla " + table);
80
```

```
81              command[1] = database + "/" + table;
82
83              /*
84               * Se agrega password de conexion a BD
85               * TODO anadir password obtenida desde archivo
86               */
87              command[2] = "p4n4lyt1c5";
88
89              logger.info("Ejecutando descarga de datos.");
90
91              retVal1 = UsefulFunctions.executeCommandLine(command);
92
93              logger.debug("Resultado de la descarga : " + retVal1);
94
95              if (retVal1 == 0) {
96                  /*
97                   * Se mueve resultado a directorio compartido. Se hace con
                         un rsync desde
98                   * /var/tmp/dump_(database) a /nfs/replication, con el fin
                         de
99                   * solo copiar lo nuevo
100                  */
101                 String[] command_sync = new String[4];
102
103                 command_sync[0] = "rsync";
104                 command_sync[1] = "-vaP";
105                 command_sync[2] = "/var/tmp/dump_" + database;
106                 command_sync[3] = "/nfs/replication/.";
107
108                 logger.info("Sincronizando carpetas.");
109
110                 retVal2 = UsefulFunctions.executeCommandLine(command_sync)
                        ;
111
112                 logger.debug("Resultado de la sincronizacion : " + retVal2
                        );
113             }
114         } catch (Exception e) {
115             logger.error("Error en proceso de descarga de datos. Detalles:
                    " + e);
116         } finally {
117             return retVal1 + retVal2;
118         }
119     }
120
121     /**
122      * Metodo que ejecuta proceso de copiado desde directorio compartido
            en red,
123      * recreacion de tablas y subida de datos.
124      *
125      * @param database la base de datos sobre la cual operar
126      * @param tables listado de tablas a cargar como lista de objetos
            Table
127      * @return 0 en caso de exito, >0 en caso de error
```

```
128         * @throws RemoteException
129         */
130        @Override
131        public int executeDecompressLoadProcess(String database, String table)
               throws RemoteException {
132            int retVal1 = 1;
133            int retVal2 = 1;
134
135            logger.info("Ejecutando carga de datos");
136
137            try {
138                logger.info("Recibida info para procesar base de datos " +
                       database + " y tabla " + table);
139
140                /*
141                 * Se mueve el dump desde /nfs/replication a /var/tmp con
                        rsync
142                 */
143
144                String[] command_sync = new String[4];
145
146                command_sync[0] = "rsync";
147                command_sync[1] = "-vaP";
148                command_sync[2] = "/nfs/replication/dump_" + database;
149                command_sync[3] = "/var/tmp/.";
150
151                logger.info("Sincronizando carpetas.");
152
153                retVal1 = UsefulFunctions.executeCommandLine(command_sync);
154                logger.debug("Resultado de la sincronizacion : " + retVal1);
155
156                /*
157                 * Se recrea la tabla en la BD cliente
158                 *
159                 * Comando: mysql-ib -uroot -pp4n4lyt1c5 alvi <
                        schema_no_charset.sql
160                 */
161                logger.info("Recreando estructura de  " + table);
162                String[] command_recreate = new String[3];
163                command_recreate[0] = "scripts/recreate_table.sh";
164                command_recreate[1] = database;
165                command_recreate[2] = table;
166
167                retVal1 = UsefulFunctions.executeCommandLine(command_recreate)
                       ;
168
169                if (retVal1 == 0) {
170                    /*
171                     * Se ejecuta script de subida de datos
172                     */
173                    logger.info("Subiendo datos de " + table);
174                    String[] command_restore = new String[2];
175                    command_restore[0] = "sh";
176                    command_restore[1] = "/var/tmp/dump_" + database
```

```
177                         + "/" + table + "/restore.sh";
178
179                 retVal2 = UsefulFunctions.executeCommandLine(
                        command_restore);
180
181                 logger.debug("Resultado de la carga : " + retVal2);
182                 //logger.info("Tabla correctamente cargada");
183                 logger.info("Fin de proceso de subida de datos");
184             }
185         } catch (Exception e) {
186             logger.error("Error en proceso de descarga de datos. Detalles:
                    " + e);
187         } finally {
188             return retVal1 + retVal2;
189         }
190     }
191 }
```

B .3. Script de descarga de datos dump_ib_db.sh

```
1  #!/bin/bash
2  # Script de apoyo a trasvasije de bases de datos infobright
3  # (tambien mysql tablas myisam).
4  # Penta Analytics S.A.
5  #
6  # Ejemplo de uso:
7  #   ./dump_ib_db.sh nombre_de_la_bd password_conexion_root
8  #   ./dump_ib_db.sh nombre_de_la_bd password_conexion_root /otro/lugar
9  #   ./dump_ib_db.sh 'nombre_de_la_bd/tabla1 tabla2 tabla3' password_root
10 #
11 #   para restaurar:
12 #   mysql-ib -u root -p < schema.sql
13 #   bash restore.sh
14 #
15 # OBS. 1: corregir restore.sh en caso de cambio de password.
16 # OBS. 2: tambien se puede utilizar schema_no_charset.sql para el caso de
17 #         no se desee que el database y las tablas tengan el charset
       original
18 #         si no que el por default al momento de restaurar.
19 # OBS. 3: El usuario a ejecutar este script debe pertener al grupo mysql,
20 #         esto debido a que el archivo .dump pertenece a este usuario y
21 #         al no pertener al grupo, el archivo no se puede comprimir.
22 # OBS. 4: Para tablas de mas de 250000000 filas se generaran varios
       archivos
23 #         de dump para la misma tabla. Cambiar este limite con la variable
24 #         MAXROWS.
25 MAXROWS=50000000
26 BASEDEST='/var/tmp'
27 DATABASE=${1%%/*}
28
29 echo "DEBUG: Variable database: $DATABASE"
```

```
30
31  if [ "$1" != "$DATABASE" ] ; then
32    TABLES=${1#*/}
33  fi
34
35  echo "DEBUG: Variable tables: $TABLES"
36
37  echo "BD a procesar: '$DATABASE'."
38  if [ "" != "$TABLES" ] ; then
39    echo "Tablas a procesar: '$TABLES'."
40  fi
41  if [ "" != "$3" ] ; then
42    if [ -d "$3" ] ; then
43      BASEDEST="$3"
44    else
45      echo ""
46      echo "ERROR: no existe directorio base '$3'."
47      exit 1;
48    fi
49  fi
50
51  # Lo que estaba antes con el script de Edgard
52
53  DEST="$BASEDEST/dump_$DATABASE"
54  echo "Destino: '$DEST'."
55
56  if [ ! -e "$DEST" ] ; then
57      echo ""
58      echo "No existe '$DEST'. Se creara este directorio."
59      mkdir -v "$DEST"
60      chmod -v 777 "$DEST";
61  fi
62
63  echo "Borrando contenido previo de directorio $DEST."
64  rm -rvf "$DEST/tablas.txt"
65  rm -rvf "$DEST/tablas_todas.txt"
66  rm -rvf "$DEST/schema_no_charset.sql"
67  rm -rvf "$DEST/schema.sql"
68
69
70  if [ "" != "$TABLES" ] ; then
71    /usr/local/infobright/bin/mysqldump -u root -p$2 -S /tmp/mysql-ib.sock
         --protocol=SOCKET --compact --single-transaction -d $DATABASE $TABLES
          > $DEST/schema.sql
72  else
73    /usr/local/infobright/bin/mysqldump -u root -p$2 -S /tmp/mysql-ib.sock
         --protocol=SOCKET --compact --single-transaction -d --single-
         transaction -B $DATABASE > $DEST/schema.sql
74  fi
75  chmod 777 "$DEST/schema.sql"
76  cat "$DEST/schema.sql" | sed 's/\/\*[^\*]*\*\///g' | sed 's/BRIGHTHOUSE
       .*;/BRIGHTHOUSE;/'| sed 's/MyISAM.*;/MyISAM;/' | sed 's/COLLATE [^ ]*//
       ' | sed 's/CHARACTER SET [^ ,]*//' > "$DEST/schema_no_charset.sql"
77  chmod 777 "$DEST/schema_no_charset.sql"
```

```
78  echo "show table status;" | mysql-ib -uroot -p$2 -S /tmp/mysql-ib.sock --
        protocol=SOCKET $DATABASE | awk '{print $1 " " $2}' | tail --lines=+2 >
         "$DEST/tablas_todas.txt"
79  if [ "" != "$TABLES" ] ; then
80    touch "$DEST/tablas.txt"
81    for tablename in $TABLES; do
82      cat "$DEST/tablas_todas.txt" | grep -w "$tablename" >> "$DEST/tablas.
            txt"
83    done
84  else
85    cp "$DEST/tablas_todas.txt" "$DEST/tablas.txt"
86  fi
87  chmod 777 "$DEST/tablas.txt"
88  echo -n "Cantidad de tablas: "
89  cat "$DEST/tablas.txt" | wc -l
90
91  IFS=$'\n'
92  for line in $( cat $DEST/tablas.txt ); do
93    table=${line% *};
94    type=${line#* };
95    echo -n "Obteniendo datos tabla '$table' ($type)..."
96
97    # Chequear primero tamano de tabla para optimizar dump
98    ROWS=`mysql-ib  -uroot -p$2 -sN -e "SELECT COUNT(*) FROM $table;"
          $DATABASE`
99    ROWS="${ROWS%\\n}"
100
101
102   OFFSET=0
103   while [ $OFFSET -le $ROWS ]
104   do
105                 if [ ! -e "$DEST/$table" ] ; then
106                     echo ""
107                     echo "No existe directorio $DEST/$table, creando..."
108                 else
109                     echo ""
110                     echo "Ya existe directorio $DEST/$table, borrando..."
111                     rm -fvr "$DEST/$table"
112                     echo "Limpieza completa.";
113                 fi
114
115                 mkdir -v "$DEST/$table";
116
117                 chmod -v 777 "$DEST/$table";
118                 touch "$DEST/$table/restore.sh"
119                 chmod 777 "$DEST/$table/restore.sh"
120     DUMPFILENAME="$DEST/$table/$OFFSET.dump"
121
122                 /usr/local/infobright/bin/mysqldump -u root -p$2 -S  /tmp/
                        mysql-ib.sock --protocol=SOCKET --compact --add-drop-
                        table --single-transaction  -d $DATABASE $table > $DEST
                        /$table/schema.sql
123                 #/usr/local/infobright/bin/mysqldump -u root -p$2 -S  /tmp
                        /mysql-ib.sock --protocol=SOCKET --compact --add-drop-
```

```
                    table -d $DATABASE $table > $DEST/$table/schema.sql
124
125                 # En caso de que la descarga de datos falle no proseguimos
126                 if [ "$?" -ne 0 ] ; then
127                     echo "Fallo la descarga. Abortar mision";
128                     exit 1;
129                 fi
130
131                 chmod 777 "$DEST/$table/schema.sql"
132                 cat "$DEST/$table/schema.sql" | sed 's/\/\*[^\*]*\*\///g'
                    | sed 's/BRIGHTHOUSE.*;/BRIGHTHOUSE;/'| sed 's/MyISAM
                    .*;/MyISAM;/' | sed 's/COLLATE [^ ]*//' | sed 's/
                    CHARACTER SET [^ ,]*//' > "$DEST/$table/
                    schema_no_charset.sql"
133                 chmod 777 "$DEST/$table/schema_no_charset.sql"
134
135     if [ "$type" = "BRIGHTHOUSE" ] ; then
136       # Intentar "dumpear" hasta que resulte...
137       # Este loop es para casos extranos en caso de que salga algun error
138       NOTOK=1
139       while [ $NOTOK -eq 1 ]
140       do
141         mysql-ib -uroot -p$2 -e "SET @bh_dataformat = 'txt_variable';
              SELECT * FROM $table LIMIT $MAXROWS OFFSET $OFFSET INTO OUTFILE
               '$DUMPFILENAME' FIELDS TERMINATED BY '\t' ENCLOSED BY 'NULL'
              LINES TERMINATED BY '\n';" $DATABASE
142         NOTOK=$?
143       done
144     else
145       mysql-ib -uroot -p$2 -e "SET @bh_dataformat = 'txt_variable'; SELECT
              * FROM $table LIMIT $MAXROWS OFFSET $OFFSET INTO OUTFILE '
            $DUMPFILENAME' FIELDS TERMINATED BY '\t' ENCLOSED BY '' LINES
            TERMINATED BY '\n';" $DATABASE
146     fi
147     echo -n " OK."
148     if [ $ROWS -gt $MAXROWS ] ; then
149       echo -n " OFFSET $OFFSET."
150     fi
151     echo -n " Comprimiendo: "
152     # Intentar ejecutar lzma hasta que resulte...
153     NOTOK=1
154     while [ $NOTOK -eq 1 ]
155     do
156       lzma -1v "$DUMPFILENAME" 2> /dev/null
157       NOTOK=$?
158     done
159     echo -n " ..."
160     echo "date +'%Y-%m-%d %H:%M:%S %z'" >> "$DEST/$table/restore.sh"
161     echo "mkfifo --mode=0666 \"/tmp/namedPipe_$table\"" >> "$DEST/$table/
          restore.sh"
162     echo "lzma -dc \"$DUMPFILENAME.lzma\" > \"/tmp/namedPipe_$table\" &"
          >> "$DEST/$table/restore.sh"
163     if [ "$type" = "BRIGHTHOUSE" ] ; then
```

```
164         echo "mysql-ib -uroot -p$2 --show-warnings -v -v -e \"SET
               @bh_dataformat = 'txt_variable'; LOAD DATA INFILE '/tmp/
               namedPipe_$table' INTO TABLE $table FIELDS TERMINATED BY '\t'
               ENCLOSED BY 'NULL';\" $DATABASE" >> "$DEST/$table/restore.sh"
165       else
166         echo "mysql-ib -uroot -p$2 --show-warnings -v -v -e \"SET
               @bh_dataformat = 'txt_variable'; LOAD DATA INFILE '/tmp/
               namedPipe_$table' INTO TABLE $table FIELDS TERMINATED BY '\t'
               ENCLOSED BY '';\" $DATABASE" >> "$DEST/$table/restore.sh"
167       fi
168       echo "rm \"/tmp/namedPipe_$table\"" >> "$DEST/$table/restore.sh"
169       echo "date +'%Y-%m-%d %H:%M:%S %z'" >> "$DEST/$table/restore.sh"
170       echo " OK."
171       OFFSET=$(( OFFSET + MAXROWS ))
172     done
173   done
```

C . Script MySQL-Proxy

```
1   --[[ $%BEGINLICENSE%$
2   Copyright (c) 2007, 2009, Oracle and/or its affiliates. All rights
       reserved.
3
4   This program is free software; you can redistribute it and/or
5   modify it under the terms of the GNU General Public License as
6   published by the Free Software Foundation; version 2 of the
7   License.
8
9   This program is distributed in the hope that it will be useful,
10  but WITHOUT ANY WARRANTY; without even the implied warranty of
11  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12  GNU General Public License for more details.
13
14  You should have received a copy of the GNU General Public License
15  along with this program; if not, write to the Free Software
16  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
17  02110-1301  USA
18
19  $%ENDLICENSE%$ --]]
20  --[[
21
22
23
24  --]]
25  require "luasql.mysql"
26  local tokenizer = require("proxy.tokenizer")
27  local lb        = require("proxy.balance")
28  local auto_config = require("proxy.auto-config")
29  local commands    = require("proxy.commands")
30  local requiero_commit = 0
31
```

```
32  -- pool de conexiones
33  --[[
34  local min_idle_connections = 4
35  local max_idle_connections = 8
36  --]]
37  if not proxy.global.config.rwsplit then
38    proxy.global.config.rwsplit = {
39      min_idle_connections = 4,
40      max_idle_connections = 8
41    }
42  end
43
44  -- track de transacciones
45  local is_in_transaction = false
46
47
48  -- guardo el resultado del load data
49  local global_statement = ""
50
51  -- variable para indicar de que operacion proviene la instruccion
52  local global_which_operation = 0
53
54  function connect_server ()
55    print ("[connect_server]")
56
57    local rw_ndx = 0
58
59    for i=1, #proxy.global.backends do
60      local s = proxy.global.backends[i]
61      local pool = s.pool
62      local cur_idle = pool.users[""].cur_idle_connections
63
64      pool.min_idle_connections = proxy.global.config.rwsplit.
          min_idle_connections
65      pool.max_idle_connections = proxy.global.config.rwsplit.
          max_idle_connections
66
67      if s.state ~= proxy.BACKEND_STATE_DOWN then
68        if s.type == proxy.BACKEND_TYPE_RW and
69          cur_idle < pool.min_idle_connections then
70          proxy.connection.backend_ndx = i
71          break
72        elseif s.type == proxy.BACKEND_TYPE_RO and
73          cur_idle < pool.min_idle_connections then
74          proxy.connection.backend_ndx = i
75          break
76        elseif s.type == proxy.BACKEND_TYPE_RW and
77          rw_ndx == 0 then
78          rw_ndx = i
79        end
80      end
81    end
82
83    if proxy.connection.backend_ndx == 0 then
```

```
84      proxy.connection.backend_ndx = rw_ndx
85    end
86
87    if proxy.connection.server then
88      return proxy.PROXY_IGNORE_RESULT
89    end
90
91    print ("Escogiendo finalmente a : " .. proxy.connection.backend_ndx)
92  end
93
94  function read_auth_result (auth)
95    if auth.packet:byte() == proxy.MYSQLD_PACKET_OK then
96      proxy.connection.backend_ndx = 0
97    elseif auth.packet:byte() == proxy.MYSQLD_PACKET_EOF then
98    elseif auth.packet:byte() == proxy.MYSQLD_PACKET_ERR then
99    end
100 end
101
102 function read_query(packet)
103   local c = proxy.connection.client
104   local is_insert = 0
105   local is_load = 0
106   local is_update = 0
107   local is_delete = 0
108   local is_select = 0
109   local is_truncate = 0
110   local is_droptable = 0
111   local is_create = 0
112   local is_not_supported = 0
113   local where_statement = ""
114   print("[read_query] " .. proxy.connection.client.src.name)
115   print("  current backend   = " .. proxy.connection.backend_ndx)
116   print("  client default db = " .. c.default_db)
117   print("  client username   = " .. c.username)
118
119   if packet:byte() == proxy.COM_QUIT then
120     -- don't send COM_QUIT to the backend. We manage the connection
121     -- in all aspects.
122     print ("Llego un quit!")
123     if requiero_commit == 1
124     then
125       print ("y falta un commit")
126       print ("El statement es: " .. global_statement)
127       proxy.queries:prepend(3, string.char(proxy.COM_QUERY) .. "COMMIT", {
            resultset_is_needed=true})
128
129       if global_which_operation == 1
130       then -- Se trata de un drop_table o un load table
131         proxy.queries:prepend(2, string.char(proxy.COM_QUERY) .. "UPDATE
              Replication.r_tables rt SET status=-1, modified_time=NOW() " ..
               global_statement, {resultset_is_needed=true})
132
133       elseif  global_which_operation == 2
134       then -- Se trata de un create_table
```

```
135           for i=1, #proxy.global.backends do
136             local s = proxy.global.backends[i]
137             if s.type == proxy.BACKEND_TYPE_RO
138             then
139               proxy.queries:prepend(2, string.char(proxy.COM_QUERY) .. "
                      INSERT INTO Replication.r_tables VALUES (" ..
                      global_statement .. ", '".. s.dst.address .. "', -1, NOW())
                      ;"
140               , {resultset_is_needed=true})
141             end
142           end
143         elseif  global_which_operation == 3
144         then -- Se trata de un drop_table
145           proxy.queries:prepend(2, string.char(proxy.COM_QUERY) .. "DELETE
                 FROM Replication.r_tables rt " .. global_statement, {
                 resultset_is_needed=true})
146         end
147         return proxy.PROXY_SEND_QUERY
148       else
149         proxy.response = {
150           type = proxy.MYSQLD_PACKET_OK,
151         }
152         return proxy.PROXY_SEND_RESULT
153       end
154     end
155
156
157
158     --if not is_in_transaction and packet:byte() == proxy.COM_QUERY then
159     if packet:byte() == proxy.COM_QUERY then
160       local tokens = tokenizer.tokenize(packet:sub(2))
161       local print_tables = ""
162
163       -- just for debug
164       for i = 1, #tokens do
165         -- print the token and what we know about it
166         local token = tokens[i]
167         local txt = token["text"]
168
169         if token["token_name"] == 'TK_STRING' then
170           txt = string.format("%q", txt)
171         end
172         print(i .. ": " .. " { " .. token["token_name"] .. ", " .. txt .. "
              }" )
173
174         if token["token_name"] == 'TK_SQL_LOAD' then
175           is_load = is_load + 1
176           break
177         elseif token["token_name"] == 'TK_SQL_UPDATE' then
178           is_update = is_update + 1
179           break
180         elseif token["token_name"] == 'TK_SQL_INSERT' then
181           is_insert = is_insert + 1
182           break
```

```
183         elseif token["token_name"] == 'TK_SQL_DELETE' then
184           is_delete = is_delete + 1
185           break
186         elseif token["token_name"] == 'TK_SQL_SELECT' then
187           is_select = is_select + 1
188           break
189         elseif token["token_name"] == 'TK_SQL_DROP' then
190           is_droptable = is_droptable + 1
191           break
192         elseif token["token_name"] == 'TK_SQL_CREATE' then
193           is_create = is_create + 1
194           break
195         elseif txt:lower() == 'truncate' then
196           is_truncate = is_truncate + 1
197           break
198         elseif txt:lower() == 'temporary' then
199           is_create = 0
200           is_droptable = 0
201           break
202         elseif token["token_name"] == 'TK_SQL_CALL'
203         or     token["token_name"] == 'TK_SQL_REPLACE'
204         or     token["token_name"] == 'TK_SQL_ALTER'
205         or     token["token_name"] == 'TK_SQL_RENAME'
206         then
207           is_not_supported = is_not_supported + 1
208           break
209         end
210       end
211
212       if is_update == 1 then
213         print_tables = extract_db_update(c.default_db, tokens)
214       elseif is_insert == 1 then
215         print_tables = extract_db_insert(c.default_db, tokens)
216       elseif is_delete == 1 then
217         print_tables = extract_db_delete(c.default_db, tokens)
218       elseif is_load == 1 then
219         print_tables = extract_db_load(c.default_db, tokens)
220       elseif is_load == 1 then
221         print_tables = extract_db_load(c.default_db, tokens)
222       elseif is_truncate == 1 then
223         print_tables = extract_db_truncate(c.default_db, tokens)
224       elseif is_droptable == 1 then
225         print_tables = extract_db_droptable(c.default_db, tokens)
226       elseif is_create == 1 then
227         print_tables = extract_db_create(c.default_db, tokens)
228       elseif is_not_supported == 1 then
229         -- Enviar mensaje de operacion no soportada
230         proxy.queries:reset()
231         proxy.response = {
232           type = proxy.MYSQLD_PACKET_ERR,
233           errmsg = "Operacion no soportada en replicador"
234         }
235         requiero_commit = 0
236         return proxy.PROXY_SEND_RESULT
```

```
237       end
238
239       if is_load + is_update + is_insert + is_delete + is_truncate +
              is_droptable + is_create > 0 then
240         if is_create == 1 then
241           where_statement = convert_to_insert(print_tables)
242         else
243           where_statement = convert_to_where(print_tables)
244         end
245
246         print ("El where es: " .. where_statement)
247         print ("[DEBUG] Escogiendo maestro, pues se actualizaran datos")
248         proxy.connection.backend_ndx = lb.idle_failsafe_rw()
249         --proxy.connection.backend_ndx = 1
250       end
251
252       if is_select == 1 then
253         --iniciamos conexion con servidor de replicacion
254         local env = assert (luasql.mysql())
255
256         --TODO obtener estos datos de un archivo de configuracion
257         local db, err = assert (env:connect("Replication","root","p4n4lyt1c5
                ","192.168.2.206","5029"))
258         b = "SELECT DISTINCT rt.server FROM r_tables rt WHERE rt.server NOT
                IN (SELECT DISTINCT rt.server FROM r_tables rt WHERE status <> 1)
                "
259
260         local servers_ro_updated = {}
261
262         for server in rows (db, b) do
263           print (string.format("[DEBUG] ... La IP: %s", server))
264           table.insert(servers_ro_updated, server)
265         end
266
267         db:close()
268         env:close()
269
270         if #servers_ro_updated == 0 then
271           print ("No hay servidor esclavo actualizado. Escogiendo maestro
                  ...")
272           proxy.connection.backend_ndx = lb.idle_failsafe_rw()
273         else
274           for i = 1, #proxy.global.backends do
275             local s = proxy.global.backends[i]
276             print ("[DEBUG] .. Conexiones para " .. s.dst.address .. " : ")
277             print ("Clientes :" .. s.connected_clients)
278             if  s.type == proxy.BACKEND_TYPE_RO and
279               s.state ~= proxy.BACKEND_STATE_DOWN and
280               search (servers_ro_updated, s.dst.address) and
281               s.pool.users[""].cur_idle_connections > 0
282             then
283                 print ("Escogiendo cliente " .. s.dst.address)
284                 proxy.connection.backend_ndx = i
285                 break
```

```
286             end
287           end
288         end
289       end
290
291       print("normalized query: " .. tokenizer.normalize(tokens))
292       print("")
293     end
294
295     print ("is_insert es : " .. is_insert)
296     print ("La consulta aca es : " .. packet:sub(2))
297     print ("Requiero commit : " .. requiero_commit)
298
299
300     if proxy.connection.backend_ndx == 0 then
301       -- Aun sin backend? Enviar al maestro
302       proxy.connection.backend_ndx = lb.idle_failsafe_rw()
303     end
304
305
306     if is_load + is_update + is_insert + is_delete + is_truncate +
          is_droptable + is_create > 0 then
307       print ("[DEBUG] Aniadir hack de consulta")
308
309       if is_load == 1 or is_droptable == 1 or is_create == 1 then
310         requiero_commit = 1
311         if is_load == 1 then
312           global_which_operation = 1
313         elseif is_create == 1 then
314           global_which_operation = 2
315         elseif is_droptable == 1 then
316           global_which_operation = 3
317         end
318       end
319
320       -- Si no requiero un commit posterior entonces aniado estas
             instrucciones
321       -- al final de la cola
322       if requiero_commit ~= 1 then
323         print ("aniadiendo commit a la cola")
324         proxy.queries:prepend(3, string.char(proxy.COM_QUERY) .. "COMMIT", {
              resultset_is_needed=true})
325         proxy.queries:prepend(2, string.char(proxy.COM_QUERY) .. "UPDATE
              Replication.r_tables rt SET status=-1, modified_time=NOW() " ..
              where_statement, {resultset_is_needed=true})
326       end
327
328       proxy.queries:prepend(1, packet, {resultset_is_needed=true})
329       proxy.queries:prepend(2, string.char(proxy.COM_QUERY) .. "SET
             AUTOCOMMIT=0", {resultset_is_needed=true})
330       proxy.queries:prepend(2, string.char(proxy.COM_QUERY) .. "START
             TRANSACTION", {resultset_is_needed=true})
331       global_statement = where_statement
332     else
```

```
333       print ("[DEBUG] Envio consulta directamente al servidor")
334       proxy.queries:append(1, packet, {resultset_is_needed=true})
335     end
336
337     if proxy.connection.server then
338       local s = proxy.connection.server
339       if packet:byte() ~= proxy.COM_INIT_DB and c.default_db and
340         c.default_db ~= s.default_db then
341         print ("[DEBUG] Sincronizando...")
342         proxy.queries:prepend(4, string.char(proxy.COM_INIT_DB) .. c.
              default_db, {resultset_is_needed = true})
343       end
344     end
345     print ("[DEBUG] Enviando finalmente la consulta a " .. proxy.connection.
          backend_ndx)
346
347     return proxy.PROXY_SEND_QUERY
348   end
349
350   function read_query_result (inj)
351     print ("[DEBUG] [query_result] Leyendo resultados!")
352     print ("[DEBUG] [query_result] Query: " .. inj.query:sub(2))
353     print ("[DEBUG] [query_result] Id: " .. inj.id)
354     local res      = assert(inj.resultset)
355     local flags    = res.flags
356     if inj.id ~= 1 then
357       print ("[DEBUG] [query_result] Leyendo resultados malignos! Ignorar!")
358       if inj.id == 3 then
359         print ("[DEBUG] [query_result] Llego la malicia!")
360         if requiero_commit == 1 then
361           proxy.response = {
362             type = proxy.MYSQLD_PACKET_OK,
363           }
364           requiero_commit = 0
365           global_which_operation = 0
366           return proxy.PROXY_SEND_RESULT
367         end
368         return proxy.PROXY_IGNORE_RESULT
369       elseif inj.id == 4 then
370         print ("[DEBUG] [query_result] Llego la sincronizacion")
371         if res.query_status == proxy.MYSQLD_PACKET_ERR then
372           proxy.queries:reset()
373           proxy.response = {
374             type = proxy.MYSQLD_PACKET_ERR,
375             errmsg = "No se puede cambiar a base de datos en esclavo"
376           }
377           return proxy.PROXY_SEND_RESULT
378         end
379       else
380         return proxy.PROXY_IGNORE_RESULT
381       end
382     end
383
384     if res.query_status == proxy.MYSQLD_PACKET_ERR then
```

```
385       proxy.queries:reset()
386       proxy.response = {
387         type = proxy.MYSQLD_PACKET_ERR,
388         errmsg = "Error en ejecucion"
389       }
390       requiero_commit = 0
391       return proxy.PROXY_SEND_RESULT
392     end
393
394     is_in_transaction = flags.in_trans
395     if not is_in_transaction and requiero_commit ~= 1 then
396       proxy.connection.backend_ndx = 0
397     end
398     print ("[DEBUG] [query_result] Retornando consulta sin modificar")
399   end
400
401   function disconnect_client ()
402     proxy.connection.backend_ndx = 0
403   end
404
405   function rows (connection, sql_statement)
406     local cursor = assert (connection:execute (sql_statement))
407     return function ()
408       return cursor:fetch()
409     end
410   end
411
412   function search (table, value)
413     for _,v in pairs(table) do
414       if v == value then
415         return true
416       end
417     end
418     return false
419   end
420
421   function extract_db_truncate (default_db, tokens_query)
422     local list_tables = ""
423     local db_found = 0
424     local last_literal = ""
425     local db = ""
426     for i = 1, #tokens_query do
427       local token = tokens_query[i]
428       local txt = token["text"]
429
430       if txt:lower() == 'truncate'
431         or token["token_name"] == 'TK_SQL_TABLE' then
432         -- do nothing
433       elseif token["token_name"] == 'TK_DOT' then
434         db_found = 1
435         db = last_literal
436       elseif token["token_name"] == 'TK_LITERAL' then
437         last_literal = txt
438         if db_found == 1 then
```

```
439           list_tables = db .. "." .. last_literal
440           return list_tables
441         end
442       end
443     end
444     return default_db .. "." .. last_literal
445   end
446
447   function extract_db_create (default_db, tokens_query)
448     local list_tables = ""
449     local db_found = 0
450     local last_literal = ""
451     local db = ""
452     for i = 1, #tokens_query do
453       local token = tokens_query[i]
454       local txt = token["text"]
455
456       if token["token_name"] == 'TK_SQL_CREATE'
457         or token["token_name"] == 'TK_SQL_IF'
458         or token["token_name"] == 'TK_SQL_NOT'
459         or token["token_name"] == 'TK_SQL_EXISTS'
460         or token["token_name"] == 'TK_SQL_TABLE' then
461         -- do nothing
462       elseif token["token_name"] == 'TK_DOT' then
463         db_found = 1
464         db = last_literal
465
466       elseif token["token_name"] == 'TK_LITERAL' or token["token_name"] == '
             TK_FUNCTION'  then
467         last_literal = txt
468         if db_found == 1 then
469           print ("Tratando de finalizar con :" .. db .. "." .. last_literal)
470           list_tables = db .. "." .. last_literal
471           return list_tables
472         end
473
474       elseif token["token_name"] == 'TK_OBRACE'
475         or token["token_name"] == 'TK_SQL_SELECT'
476         or token["token_name"] == 'TK_SQL_LIKE' then
477         list_tables = default_db .. "." .. last_literal
478         break
479       end
480     end
481     return list_tables
482   end
483
484   function extract_db_droptable (default_db, tokens_query)
485     local list_tables = ""
486     local db_found = 0
487     local last_literal = ""
488     local db = ""
489     local literal_at_last = 1
490
491     for i = 1, #tokens_query do
```

```
492       local token = tokens_query[i]
493       local txt = token["text"]
494
495       if txt:lower() == 'temporary'
496         or token["token_name"] == 'TK_SQL_DROP'
497         or token["token_name"] == 'TK_SQL_TABLE'
498         or token["token_name"] == 'TK_SQL_IF'
499         or token["token_name"] == 'TK_SQL_EXISTS'
500       then
501           -- do nothing
502
503       elseif token["token_name"] == 'TK_DOT' then
504           db_found = 1
505           db = last_literal
506
507       elseif token["token_name"] == 'TK_LITERAL' then
508           last_literal = txt
509
510       elseif token["token_name"] == 'TK_COMMA' then
511           if db_found == 0 then
512             db = default_db
513           end
514
515           list_tables = list_tables .. db .. "." .. last_literal .. ","
516           db = ""
517           last_literal = ""
518           db_found=0
519
520       elseif token["token_name"] == 'TK_SQL_RESTRICT'
521           or token["token_name"] == 'TK_SQL_CASCADE' then
522           literal_at_last = 0
523           if db_found == 0 then
524             db = default_db
525           end
526
527           list_tables = list_tables .. db .. "." .. last_literal .. ","
528           break;
529
530       end
531     end
532
533     if literal_at_last == 1 then
534       if db_found == 0 then
535           db = default_db
536       end
537
538       list_tables = list_tables .. db .. "." .. last_literal .. ","
539     end
540
541     return string.sub(list_tables, 1, string.len(list_tables) - 1)
542   end
543
544
545   function extract_db_update (default_db, tokens_query)
```

```
546     local list_tables = ""
547     local db_found = 0
548     local last_literal = ""
549     local db = ""
550 
551     for i = 1, #tokens_query do
552       local token = tokens_query[i]
553       local txt = token["text"]
554 
555       if token["token_name"] == 'TK_SQL_SET'
556         or token["token_name"] == 'TK_SQL_LEFT'
557         or token["token_name"] == 'TK_SQL_RIGHT'
558         or token["token_name"] == 'TK_SQL_INNER'
559         or token["token_name"] == 'TK_SQL_OUTER' then
560         if db_found == 0 then
561           db = default_db
562         end
563 
564         list_tables = list_tables .. db .. "." .. last_literal .. ","
565         db = ""
566         last_literal = ""
567         db_found=0
568         break
569 
570       elseif token["token_name"] == 'TK_SQL_UPDATE'
571         or token["token_name"] == 'TK_SQL_LOW_PRIORITY'
572         or token["token_name"] == 'TK_SQL_IGNORE' then
573         -- do nothing
574 
575       elseif token["token_name"] == 'TK_LITERAL' then
576         last_literal = txt
577 
578       elseif token["token_name"] == 'TK_DOT' then
579         db_found = 1
580         db = last_literal
581 
582       elseif token["token_name"] == 'TK_COMMA' then
583         if db_found == 0 then
584           db = default_db
585         end
586 
587         list_tables = list_tables .. db .. "." .. last_literal .. ","
588         db = ""
589         last_literal = ""
590         db_found=0
591 
592       end
593     end
594     return string.sub(list_tables, 1, string.len(list_tables) - 1)
595   end
596 
597   function extract_db_insert (default_db, tokens_query)
598     local list_tables = ""
599     local db_found = 0
```

```
600   local last_literal = ""
601   local db = ""
602
603   for i = 1, #tokens_query do
604     local token = tokens_query[i]
605     local txt = token["text"]
606     if  token["token_name"] == 'TK_OBRACE'
607       or token["token_name"] == 'TK_SQL_VALUE'
608       or token["token_name"] == 'TK_SQL_VALUES'
609       or token["token_name"] == 'TK_SQL_SET'
610       or token["token_name"] == 'TK_SQL_SELECT'
611     then
612       if db_found == 0 then
613         db = default_db
614       end
615       list_tables = list_tables .. db .. "." .. last_literal .. ","
616       db = ""
617       last_literal = ""
618       db_found=0
619
620       break
621     else
622       if  token["token_name"] == 'TK_SQL_INSERT'
623         or token["token_name"] == 'TK_SQL_LOW_PRIORITY'
624         or token["token_name"] == 'TK_SQL_DELAYED'
625         or token["token_name"] == 'TK_SQL_HIGH_PRIORITY'
626         or token["token_name"] == 'TK_SQL_IGNORE'
627       then
628           -- do nothing
629       else
630         if token["token_name"] == 'TK_LITERAL' then
631           last_literal = txt
632         else
633           if token["token_name"] == 'TK_DOT' then
634             db_found = 1
635             db = last_literal
636           end
637         end
638       end
639     end
640   end
641   return string.sub(list_tables, 1, string.len(list_tables) - 1)
642 end
643
644 function extract_db_delete (default_db, tokens_query)
645   local list_tables = ""
646   local db_found = 0
647   local last_literal = ""
648   local db = ""
649   local delete_first_type = 0
650
651   for i = 1, #tokens_query do
652     local token = tokens_query[i]
653     local txt = token["text"]
```

```
654       if  token["token_name"] == 'TK_SQL_DELETE'
655         or token["token_name"] == 'TK_SQL_LOW_PRIORITY'
656         or token["token_name"] == 'TK_SQL_QUICK'
657         or token["token_name"] == 'TK_SQL_IGNORE'
658       then
659         -- do nothing
660       else
661         if token["token_name"] == 'TK_LITERAL' then
662           if delete_first_type == 0 then
663             delete_first_type = 1
664           end
665           last_literal = txt
666         else
667           if token["token_name"] == 'TK_DOT' then
668             db_found = 1
669             db = last_literal
670           end
671           if token["token_name"] == 'TK_COMMA' then
672             if db_found == 0 then
673               db = default_db
674             end
675             list_tables = list_tables .. db .. "." .. last_literal .. ","
676             db = ""
677             last_literal = ""
678             db_found=0
679           end
680         end
681       end
682
683       if token["token_name"] == 'TK_SQL_FROM' then
684         if delete_first_type == 1 then
685           if db_found == 0 then
686             db = default_db
687           end
688           list_tables = list_tables .. db .. "." .. last_literal .. ","
689           db = ""
690           last_literal = ""
691           db_found=0
692           break
693         end
694       end
695
696
697       if token["token_name"] == 'TK_SQL_WHERE'
698         or token["token_name"] == 'TK_SQL_USING'
699         or token["token_name"] == 'TK_SQL_ORDER_BY'
700       or token["token_name"] == 'TK_SQL_LIMIT'
701       then
702         if db_found == 0 then
703           db = default_db
704         end
705         list_tables = list_tables .. db .. "." .. last_literal .. ","
706         db = ""
707         last_literal = ""
```

```
708         db_found=0
709         break
710       end
711     end
712
713     return string.sub(list_tables, 1, string.len(list_tables) - 1)
714
715   end
716
717   function extract_db_load (default_db, tokens_query)
718     local list_tables = ""
719     local db_found = 0
720     local last_literal = ""
721     local db = ""
722     local save_literals = 0
723
724     for i = 1, #tokens_query do
725       local token = tokens_query[i]
726       local txt = token["text"]
727       if  token["token_name"] == 'TK_SQL_LOAD'
728         or token["token_name"] == 'TK_SQL_INFILE'
729         or token["token_name"] == 'TK_SQL_INTO'
730         or token["token_name"] == 'TK_SQL_REPLACE'
731         or token["token_name"] == 'TK_SQL_IGNORE'
732       then
733         -- do nothing
734       else
735         if token["token_name"] == 'TK_SQL_TABLE' then
736           save_literals = 1
737         end
738
739         if token["token_name"] == 'TK_LITERAL' and save_literals == 1 then
740           if  txt == 'FIELDS'
741             or txt == 'COLUMNS'
742           then
743             if db_found == 0 then
744               db = default_db
745             end
746             list_tables = list_tables .. db .. "." .. last_literal .. ","
747             db = ""
748             last_literal = ""
749             db_found=0
750             break
751           else
752             last_literal = txt
753           end
754         else
755           if token["token_name"] == 'TK_DOT' and save_literals == 1 then
756             db_found = 1
757             db = last_literal
758           end
759           if token["token_name"] == 'TK_COMMA' and save_literals == 1 then
760             if db_found == 0 then
761               db = default_db
```

```
762             end
763             list_tables = list_tables .. db .. "." .. last_literal .. ","
764             db = ""
765             last_literal = ""
766             db_found=0
767           end
768         end
769       end
770
771       if token["token_name"] == 'TK_SQL_CHARACTER'
772         or token["token_name"] == 'TK_SQL_LINES'
773         or token["token_name"] == 'TK_SQL_IGNORE'
774         or token["token_name"] == 'TK_SQL_SET'
775         or token["token_name"] == 'TK_OBRACE'
776       then
777         if db_found == 0 then
778           db = default_db
779         end
780         list_tables = list_tables .. db .. "." .. last_literal .. ","
781         db = ""
782         last_literal = ""
783         db_found=0
784         break
785       end
786     end
787
788     return string.sub(list_tables, 1, string.len(list_tables) - 1)
789
790   end
791
792   function split(str, sep)
793     return {str:match((str:gsub("[^".. sep .."]*".. sep, "([^".. sep .."]*)
          ".. sep)))}
794   end
795
796
797
798   function convert_to_where (list)
799     local where_statement = "WHERE "
800
801     if string.find(list, ",") == nil then
802       local first = 1
803       local aux = split(list .. ".", ".")
804       for i=1, #aux do
805         if first == 1 then
806           where_statement = where_statement .. "(rt.database = '" ..  aux[i
              ].. "' AND "
807           first = 0
808         else
809           where_statement = where_statement .. "rt.name = '" .. aux[i] .. "'
              ) "
810         end
811       end
812       return where_statement
```

```
813     end
814
815     local aux = split(list .. ",", ",")
816     for i=1, #aux do
817       local first = 1
818       local aux2 = split(aux[i] .. ".", ".")
819
820       for j=1, #aux2 do
821         if first == 1 then
822           where_statement = where_statement .. "(rt.database = '" ..  aux2[j
                ].. "' AND "
823           first = 0
824         else
825           where_statement = where_statement .. "rt.name = '" .. aux2[j] .. "
                ') "
826         end
827       end
828       where_statement = where_statement .. "OR "
829     end
830     return string.sub(where_statement, 1, string.len(where_statement) - 3)
831   end
832
833   -- Funcion que convierte un listado del tipo "alvi.alvi_bdm" en una
        instruccion de insert del
834   -- tipo " 'alvi', 'alvi_bdm' " (sin espacios)
835   function convert_to_insert (list)
836     local final_word = ""
837     local first = 1
838     local aux = split(list .. ".", ".")
839
840     for i=1, #aux do
841       if first == 1 then
842         final_word = "'" .. aux[i] .. "',"
843         first = 0
844
845       else
846         final_word = final_word .. "'" .. aux[i] .. "'"
847       end
848     end
849     return final_word
850   end
```