Replicación de Bases de Datos en Modalidad "maestro-esclavo", caso de estudio: Firebird SQL Server

Cherencio Guillermo Rubén[1], Perello Mario Gerardo[1], Romero Juan Carlos[1]

[1] Universidad Nacional de Lujan, Departamento de Ciencias Básicas

Abstract. Los Sistemas de Gestión de Bases de Datos (SGBD) más evolucionados permiten la programación de desencadenadores o disparadores también llamados triggers, procedimientos y funciones en lenguaje PS/SQL o Transact-SQL. En nuestro caso práctico: Firebird/Interbase® (FB) incorpora el lenguaje de programación PL/SQL, el cual permite la programación de desencadenadores (triggers) y procedimientos almacenados (stored procedures). Como parte del código PL/SQL está disponible la instrucción EXECUTE STATEMENT, la cual permite la ejecución de comandos SQL tanto en forma local como remota. Además este motor de base de datos (FB) trabaja con un algoritmo de confirmación de dos pasadas (two-phase commit, 2PC) para este comando, lo cual es suma importancia porque permite el control de las transacciones distribuidas. De esta forma es posible determinar el éxito o fracaso de la ejecución distribuida de la transacción y de esta manera implementar una replicación de tipo homogénea maestro-esclavo basado en desencadenadores (triggers) con una granularidad de replicación a nivel de tablas. El presente trabajo pretende no solo hacer una replicación maestro-esclavo con Firebird, sino el desarrollo de una aplicación que permita a través de una interface automatizar la implementación de la replicación, generando de forma automática el código y las estructuras necesarias para la replicación y también permitir la posible recuperación en caso de fallas o anomalías. Se abren futuras líneas de investigación sobre nuevas funcionalidades a incorporar a la aplicación propuesta para facilitar la implementación de bases de datos distribuidas en Firebird, fortaleciendo esta funcionalidad no disponible en este motor de base de datos. Esta aplicación está desarrollada bajo licencia LGPL.

Palabras Clave: Java Firebird Interbase 2PC Replicación SQL PL/SQL

1 Introducción

En el marco del proyecto de investigación sobre la replicación de datos distribuidos denominado *"Replicación maestro-esclavo en bases de datos relaciones no distribuidas",* usando motores de bases de datos de libre distribución que no incorporan esta posibilidad seleccionamos el caso de Firebird/Interbase®. Este Sistema Gestor de Base de Datos[1] (SGBD) está basado en el modelo relacional [2], el cual posee una arquitectura de tipo cliente-servidor y está disponible para diferentes

plataformas. En nuestro caso haremos replicación tanto en plataforma Windows® como en Linux. Posee una serie de librerías (API) que ha permitido el desarrollo de *drivers (controladores de acceso)* para distintos entornos de desarrollo (lenguajes), dando la posibilidad de realizar aplicaciones *client-server* en entornos no distribuidos. A partir de la versión 2.5 incorpora el algoritmo 2PC[3] en la ejecución de comandos de tipo EXECUTE STATEMENT[4] haciendo posible la ejecución de comandos SQL en forma remota, indicando los datos para gestionar la conexión[1] es posible ejecutar el comando SQL en otro servidor Firebird/Interbase® y luego verificar el resultado de la transacción distribuida. De esta forma, es factible la implementación de un sistema de base de datos distribuido homogéneo[2] basado en desencadenadores (*triggers*) que permita la réplica de tablas en varios servidores Firebird/Interbase® bajo la modalidad del tipo maestro-esclavo. Es importante destacar que existen otras posibilidades de replicación, no obstante, el presente trabajo solo se limita a este enfoque de replicación.

Si bien existen múltiples soluciones de replicación para bases de datos de código abierto, podría citarse por ejemplo a Postgresql[5] ; no obstante, éste no es el caso de Firebird/Interbase®[3] y esto ha motivado el desafío del presente trabajo y otros desarrollos para el futuros en este lineamiento.

Replicar una tabla bajo este enfoque implica el desarrollo e implementación de un desencadenador o *trigger* que tome los eventos de actualización de la tabla como ser *insert, update, delete* y replique el cambio a todos los servidores esclavos. Por ej. sobre una tabla relativamente sencilla[4], se requirieron 75 líneas de código PL/SQL, de las cuales solo 5 líneas son de código específico para dicha tabla. Esto significa que en un escenario de implementación concreta y real, se requerirían miles de líneas de código no generalizables, dependientes de cada tabla, con alta tasa de probabilidad de error.

Se ha propuesto desarrollar una aplicación que permita generar todo el código PL/SQL necesario para implementar el esquema de replicación. La aplicación se ha denominado FBJReplicator[5] y está disponible en el portal SourceForge[6] bajo licencia LGPL.

1 Indicar: Servidor, usuario, contraseña y rol. No permite indicar puerto TCP/IP de conexión.

2 Los servidores deben ser Firebird/Interbase®, el comando no permite la ejecución remota en servidores de otros fabricantes.

3 Desde la página oficial Firebird/Interbase® http://www.firebirdsql.org/en/third-party-tools/ pueden vincularse sólo 2 alternativas, ambas para Windows®, una de ellas, posiblemente discontinuada y la otra es una aplicación propietaria. Otro caso es IBReplicator https://www.ibphoenix.com/products/software/ibreplicator, posiblemente la opción más evolucionada, vinculada al grupo de desarrollo de Firebird/Interbase® y con licencia comercial.

4 Una tabla con solo dos atributos, con una clave primaria simple (no compuesta).

5 Disponible en http://sourceforge.net/projects/fbjreplicator/

6 http://www.sourceforge.net

2 Recursos del trabajo y método utilizado

El presente trabajo se llevó adelante siguiendo una serie de etapas, tomando los lineamientos generales de Blanchette[6]:

1. Requisitos
2. Implementación de forma manual de la arquitectura propuesta.
3. Diseño y desarrollo de la aplicación.
4. Implementación de la aplicación y bases de datos de ejemplo. Testing.
5. Empaquetado, publicación y distribución.

2.1 Requisitos

La aplicación propuesta tiene los siguientes requisitos[7]:

2.1.1 Multiplataforma.
2.1.2 Libre distribución bajo licencia LGPL.
2.1.3 Interfaz gráfica de usuario con soporte multilingüe.
2.1.4 Estructuras de datos de replicación externas a base de datos maestra o esclavo.
2.1.5 Generación de *scripts* de replicación y/o de instalación/desinstalación de replicación en servidor maestro y esclavos.
2.1.6 Posibilidad de *activar y desactivar* la replicación (ya sea por tabla replicada o por servidor).
2.1.7 *Log de transacciones* para dar al sistema tolerancia a *fallos* y formas de recuperación en caso de caídas de servidores esclavos.
2.1.8 Posibilidad de chequear *en línea* la replicación.

2.2 Implementación de forma manual de la arquitectura propuesta.

Si bien Firebird/Interbase® brinda buena documentación, uno de los principales problemas fue contar con información técnica detallada del comando EXECUTE STATEMENT y la implementación del algoritmo 2PC en Firebird/Interbase® 2.5. Como forma de suplir este problema, se realizaron pruebas con 3 servidores Firebird/Interbase® 2.5, generando una base de datos maestra de testing y replicando una tabla de esa base de datos maestra en los restantes servidores que actuarían como esclavos. Para este caso el código PL/SQL de replicación fue generado manualmente, tratando de escribirlo lo más genérico y reutilizable posible, para tratar de ayudar a su automatización futura.

2.3 Implementación en la base de datos maestra

Se implementaron las estructuras básicas para la tolerancia a fallos y su forma de recuperación, los desencadenadores (*triggers*) implementados en la base de datos maestra se apoyan en las siguientes relaciones:

tbl_config_server (**id**, nserver, ndb, nuser, npwd, nrole, active)

tbl_config_table (**id, ntable**, ttable, ninsert_fmt, nupdate_fmt, ndelete_fmt, active)

7 Se plantean los requisitos generales, para ser considerados en el diseño y que los mismos puedan implementarse si el proyecto cuenta con los tiempos y recursos necesarios.

tbl_log (**nid**, nserver, ntable, nsql, ndb, nuser, npwd, nrole, ntr, napply, nsqlerror, ngdscode)

La relación *tbl_config_server* tendrá una tupla por cada servidor esclavo que replique una o más tablas. La relación *tbl_config_table* tendrá una tupla por cada tabla a replicar en un determinado servidor esclavo (vinculado con la clave extranjera *id* a *tbl_config_server*). Los atributos *nserver, ndb, nuser, npwd, nrole* se refieren a la identificación del servidor Firebird/Interbase® 2.5, la base de datos, el usuario, la contraseña y el rol del usuario respectivamente. El atributo *active* permite activar y desactivar el servicio de replicación (requisito 2.1.6). Los atributos indicados con negritas y subrayados indican la clave primaria de cada relación. El atributo *ntable y ttable* se refieren a la tabla origen a replicar y tabla destino respectivamente[8]. La tabla *tbl_log* (requisito 2.1.7) guarda los datos de conexión utilizados al momento de ejecutar el comando sql remoto (*nsql)*, si el mismo fue satisfactorio o no (*napply*); en caso de error, se guardan los códigos de error que retorna el motor Firebird/Interbase® (*nsqlerror, ngdscode*; requisito 2.1.8), guarda el número de transacción (*ntr*) el cual se usará en caso de recuperación de un servidor esclavo caído.

Se debe implementar un desencadenador (*trigger*) por cada tabla a replicar, cuyo pseudocódigo para la tabla "*A"* es el siguiente:

```
inicio

config = tbl_config_server ⋈ tbl_config_table

R = σ active = 's' y ntable = 'A' (config)
para cada tupla en R hacer:
  si insertando entonces sql = ninsert_fmt
  si borrando entonces sql = ndelete_fmt
  si cambiando entonces sql = nupdate_fmt
  reemplazar en sql los valores de los atributos de la tabla 'A'
  ntr = numero de transacción actual
 ejecutar p_apply(ntr,sql) en servidor remoto
  si ejecucion ok entonces
     apply='si'
  sino
     apply='no'
     guardo códigos FB de error
  fin si
  grabo tupla en tbl_log
fin para
fin
```

8 Como ejemplo, una tabla denominada "A" podría replicarse con otra denominación "B" en un servidor esclavo.

Se deben tener los datos de configuración de la tabla en cuestión, junto con los servidores esclavos que la replican; el código SQL a ejecutar se apoya en 3 máscaras de formato del comando para hacer *insert* (*ninsert_fmt*), *update* (*nupdate_fmt*) y *delete* (*ndelete_fmt*) respectivamente[9] y se realiza una ejecución remota del procedimiento almacenado *p_apply* que recibe como argumento el código SQL a ejecutar y el número actual de transacción. Por último, se guarda el resultado de la ejecución en la tabla de log *tbl_log*.

2.4 Implementación en base de datos esclava

La tabla *tbl_tr(tr)* contiene una sola tupla que guarda el último número de transacción aplicado en la base de datos (requisito 2.1.7). Por cada tabla replicada se implementó un desencadenador (*trigger*) de *before insert, update, delete* para no permitir cambios en la tabla, a menos que se trate del usuario de replicación o bien el administrador del sistema[10]. Por último, el procedimiento *p_appy(ntr,sql)* ejecuta el comando *sql* y actualiza *tbl_tr* con *ntr* (número de transacción enviado desde base de datos maestra). De esta forma, es posible "poner al día" un servidor esclavo que estuvo fuera de servicio; a partir de los datos guardados en las relaciones *tbl_log, tbl_tr*.

2.5 Diseño y desarrollo de la aplicación.

Teniendo en cuenta los requisitos 1.1, 1.2, 1.3; junto con la disponibilidad del controlador JDBC Jaybird[11] para Firebird/Interbase®, se optó por un desarrollo en Java SE con una interfaz gráfica desarrollada en Swing[12]. Acorde con los requisitos 1.3, 1.4, el sistema utilizará archivos de propiedades para guardar en los mismos los mensajes en distintos idiomas, así como también los datos de configuración de la replicación.

Optamos por una interfaz gráfica que permita seleccionar el servidor maestro, luego seleccionar los servidores esclavos, las tablas a replicar por cada servidor esclavo; guardar todos los datos asociados a la replicación en archivos de propiedades y permitir al usuario instalar la replicación; ya sea generando un *script* de replicación o bien conectándose a los distintos servidores y ejecutando los comandos DDL[13] que correspondan.

La aplicación es orientada a objetos, posee abstracciones simples y fáciles de comprender [7] tales como las siguientes clases:

Clase	Descripción
MainView	Interfaz gráfica

9 Se debe a que no fue posible generalizar una regla de generación de código, ya que el código sql de insert, update y delete cambia acorde con los distintos tipos de datos de los atributos de la tabla.

10 El modelo propuesto de replicación, todos los servidores esclavos son sólo de consulta, no se permiten cambios en tablas replicadas.

11 Disponible su descarga en http://www.firebirdsql.org/en/jdbc-driver/

12 Framework Java para el desarrollo de interfaz gráfica.

13 Data Definition Language, lenguaje de definición de datos para la creación, destrucción y modificación de objetos dentro de un SGBD.

FileIni	Manejo de archivos de propiedades
FBCon	Manejo de conexión Firebird/Interbase®, ejecución de consultas y comandos DDL
StoredProcedure	Permite ejecución de procedimientos almacenados
SqlTable	Representación de una tabla en una base de datos Firebird/Interbase®
SqlTableColumn	Representación de una columna de una tabla en una base de datos Firebird/Interbase®
SqlTrigger	Representación de un desencadenador (*trigger*) asociado a una tabla en una base de datos Firebird/Interbase®
SwingFactory	Creación de componentes swing utilizados en interfaz gráfica
ViewHtml	Ventana para la visualización genérica de contenido html

Tabla 1. Clases proyecto FBJReplicator

El controlador JDBC Jaybird ha permitido realizar sobre Firebird/Interbase® todas las operaciones que requirió esta aplicación. Solo se encontraron dificultades en cuestiones vinculadas con la administración de los servidores[14].

2.6 Implementación de la aplicación y bases de datos de ejemplo. Testing

La aplicación fue probada sobre el mismo conjunto de bases de datos y servidores que fueron utilizados en la implementación de forma manual de la arquitectura propuesta; generando la replicación en forma automática y verificando que los resultados obtenidos eran coincidentes con la implementación manual.

2.7 Empaquetado, publicación y distribución.

La aplicación desarrollada se denomina como proyecto *"FBJReplicator"* el cual se distribuye bajo licencia LGPL[15] . Se incluyen los programas fuentes, ejemplos, documentación, etc. en una estructura de directorios comprimidos en formato *zip* que los usuarios pueden descargar, descomprimir y compilar utilizando un compilador Java 1.7 o superior sobre la carpeta principal. El software está configurado para trabajar en idioma español e inglés. La documentación está escrita en español. La distribución y publicación de este trabajo se realiza a través del portal SourceForge. Este portal también posibilitó el trabajo en forma colaborativa entre los miembros de esta investigación y colaboradores, a través de su servidor SVN[16].

3 Resultados

Hasta el momento no se han obtenido resultados de implementaciones concretas en entornos de producción, teniendo en cuenta esta como la primera versión del

14 Implementación de "hot-backups", administración de usuarios en forma remota.
15 http://www.gnu.org/licenses/lgpl.html
16 Sistema de control de versiones Apache Subversión http://subversion.apache.org/

sistema[17], todas las pruebas realizadas han sido en entornos denominados no productivos. Se usaron redes locales y pruebas en un mismo computador con servidores esclavos virtualizados. Se han realizado las pruebas tanto en servidores Firebird/Interbase® 2.5 bajo plataforma Windows® como Linux. Los resultados obtenidos son alentadores, han permitido la depuración de varios errores y se han establecido tiempos razonables de replicación para un entorno distribuido de tipo maestro-esclavo.

Se requiere de administración o bien del desarrollo de mecanismos de auto-administración, en especial, cuando se registran caídas de servidores esclavos, puede observarse la lentitud en las actualizaciones de datos en tablas replicadas debido a la imposibilidad de confirmación de la transacción a través de 2PC. Esta situación puede mejorarse fácilmente con la incorporación de un mecanismo que permita desactivar una replicación luego de, por ejemplo, una cantidad *N* fallas seguidas; de esta forma, la arquitectura podría funcionar más rápido y luego "poner al día" el servidor que nuevamente entra en servicio.

4 Discusión

Con los resultados que se obtuvieron se puede decir que los mismos alientan a pruebas en entornos de producción, no obstante, es requisito la completa implementación del sistema que incluya: *administración* de usuarios de replicación, *backup* de servidores, *activación/desactivación* de replicación (manual y/o automática), *notificaciones*, *monitoreo, borrado/depuración* de log de transacciones, etc.

Es importante considerar las limitaciones de la implementación de este modelo de replicación. Por ejemplo, no es posible replicar utilizando comandos SQL que incluyan llamadas a funciones o expresiones que podrían arrojar resultados distintos en los distintos servidores, a lo largo del tiempo en que transcurre la transacción distribuida[18]. Ante la caída o fallo de un servidor esclavo, solo es posible "ponerlo al día", es decir, hacer un *redo* a partir de la información del log de transacciones, pero no es posible hacer un *undo* de los servidores[19]; pues el log de transacciones sólo almacena los comandos SQL ejecutados y no los datos almacenados antes o después de la ejecución del comando SQL.

Comparando con los productos disponibles para PostgreSQL, alientan a futuros desarrollos en este sentido para Firebird/Interbase®, por ejemplo, la creación de un

17 Al momento de escribir este artículo, el sistema sólo ha implementado la funcionalidad mínima de replicación para los tipos de datos básicos de las columnas de las tablas Firebird/Interbase® 2.5. Se requiere contar con otras herramientas adicionales para hacer pruebas productivas, como por ejemplo, la administración de usuarios y backups de los servidores.

18 Para citar un caso, la expresión CURRENT_TIMESTAMP no podría utilizarse dentro de un comando sql de replicación, en vez de ello, debería asignarse ese valor y pasar el valor a todos los servidores esclavos.

19 En este sentido, sólo es posible hacer un *rollback* de la transacción distribuida actual (por ejemplo, porque la misma no fuese confirmada -al menos- por uno de los servidores esclavos); pero una vez confirmada, ya no sería posible su *rollback*.

servidor de tipo middleware (*statement-based replication middleware*), replicación multimaster sincrónica o asincrónica, particionado de datos, ejecución paralela de queries[5], etc.

5 Conclusión

Es apreciable una falencia en Firebird/Interbase® en cuanto a documentación y productos evolucionados y bien probados que implementen reparto y distribución de datos; no existen para Firebird/Interbase® opciones tan variadas como las ofrecidas para PostgreSQL.

La aplicación propuesta se considera un aporte a las opciones de replicación disponibles para Firebird/Interbase®, alienta a su comunidad de usuarios a la implementación de sistemas distribuidos que permitan la evolución y desarrollo de este sistema, favorecidos por el tipo de licencia elegido y la posibilidad de continuar trabajando desde Sourceforge.

Esta aplicación requiere de la implementación de herramientas adicionales para ser usado en un sistema de producción real, no obstante, los resultados obtenidos con las opciones básicas ya implementadas son muy alentadoras e indican su factibilidad de implementación en un sistema productivo real.

Agradecimientos
Al Mg. Jorge Peri, al Ing. Luis H. Perna, al Lic. Pedro S. Asis, al Ing. Bono Juan, al Ing. Colussi Diego por facilitar los medios y recursos necesarios para el armado de este proyecto.

Referencias

1. Castaño Miguel, Adoración de; Velthuis Piattini, Mario; Martinez, Esperanza Marcos, Diseño de Bases de Datos Relacionales, Editorial Ra-Ma, Madrid, 2000, ISBN 84-7897-385-0, Pag 5.
2. Silberschatz, Abraham; Korth, Henry F.; Sudarshan S., Fundamentos de Bases de Datos, 4ta.Ed., Ed. Mc Graw Hill, Madrid, 2002, ISBN: 0-07-228363-7, Pag. 17.
3. Rahimi, Saeed K.; Haug, Frank S, Distributed Database Management Systems. A practical approach, John Wiley & Sons Inc. Publication, IEEE Computer Society, 2010, ISBN: 978-0-470-40745-5, Pag. 329-345.
4. Vinkenoog, Paul et al., Firebird 2.5 Language Reference Update, Oct 8 2011, disponible en http://www.firebirdsql.org/file/documentation/reference_manuals/reference_material/html/langrefupd25.html
5. PostgreSQL 9.1 Manual, Chapter 25. High Availability, Load Balancing, and Replication, disponible en http://www.postgresql.org/docs/9.1/static/high-availability.html
6. Blanchette, Jasmin, The Little Manual of API Design, Trolltech, a Nokia company, June 19, 2008, disponible en http://www4.in.tum.de/~blanchet/api-design.pdf
7. Jackson, Daniel, Software Abstractions, MIT Press, 2006.

Datos de Contacto:
Cherencio, Guillermo Rubén. Universidad Nacional de Lujan, Departamento de Ciencias Básicas. Rutas 5 y 7, Lujan, CP 6700, Provincia de Buenos Aires, República Argentina. E-mail: grchere@yahoo.com.
Perello, Mario. Universidad Nacional de Lujan, Departamento de Ciencias Básicas. Rutas 5 y 7, Lujan, CP 6700, Provincia de Buenos Aires, República Argentina. E-mail: mperello04@gmail.com.
Romero, Juan Carlos. Universidad Nacional de Lujan, Departamento de Ciencias Básicas. Rutas 5 y 7, Lujan, CP 6700, Provincia de Buenos Aires, República Argentina. E-mail: juancarlosjromer@gmail.com.