Universidad de Alcalá
Escuela Politécnica Superior

Grado en Sistemas de Información

Trabajo Fin de Grado

Replicación multimaestro en bases de datos PostgreSQL

Autor: Alejandro Escobar Martín

Tutor/es: Iván González Diego

2017

Trabajo Fin de Grado

Replicación multimaestro en bases de datos PostgreSQL

Autor: Alejandro Escobar Martín

Tutor/es: Iván González Diego

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

FECHA: 26-09-2017

Índice

Resumen

En este proyecto se analizarán las distintas opciones a la hora de replicar una base de datos PostgreSQL, se elegirá la que ofrezca la replicación multimaestro más apropiada para el proyecto y se estudiará e implementará en una base de datos distribuida en dos host, una con PostgreSQL 8.4 instalado en un Oracle Linux y otra con PostgreSQL 9.4 alojada en una máquina Windows.

Summary

This proyect will analyce the various available options to replicate a PostgreSQL database, once analyced , the solution that offers the best multimaster replication will be studied and implemented in a 2 host distributed database enviroment One of the databases will be a PostgreSQL 8.4 running in a Oracle Linux S.O , and the other Will be a PostgreSQL 9.4 running in a Windows 10.

Palabras clave

Base de datos distribuida: Conjunto de bases de datos relacionadas lógicamente, distribuidas geográficamente.

Maestros: Nodo origen de los datos.

Esclavos: Nodo destino de los datos.

Stand-by: Copia intacta usada como backup de una base de datos.

Performance: Rendimiento de un programa o proceso.

Fail over (Conmutacion por error): Consiste en el relevo de un sistema por otro secundario en caso de fallo.

Data warehouse: Almacen de datos donde se guardan de manera segura la información de una organización.

Load balancing: Distribuye la carga que suponen los datos entre los distintos nodos de una base de datos:

Aplicaciones de misión crítica: Aplicaciones que, por su importancia, no pueden estar caídas o funcionando erróneamente.

Atomicidad: Propiedad que asegura si una transacción se ha realizado o no, pero que no se haya quedado a medias.

Commit: Se produce cuando el resultado de una transacción es válido.

WAL: Write Ahead Log es el método que utiliza PostgreSQL para registrar una transacción.

Md5: Es un algoritmo de encriptación.

Switchover: Se produce cuando se cambia el rol de una copia primaria a una secundaria y viceversa.

Cluster: Un cluster de bases de datos es colección de servidores independientes pero interconectados.

BLOB: Es un tipo de datos diseñado para almacenar datos de gran tamaño.

Workaround: Solución a un problema improvisto.

Resumen Extendido

Durante el desarrollo de este proyecto, se van a analizar las distintas opciones disponibles para replicar una base de datos PostgreSQL. Tras esto, se elegirá la que mejor se adapte a nuestra situación, estudiándola e investigándola más a fondo para poder implementarla en nuestra base de datos.

Para la consecución del proyecto, se crearán dos entornos, uno de pruebas sobre el que se realizaran las distintas pruebas y otro de producción, que será el mostrado en la defensa.

En ambos entornos, se utilizarán las mismas herramientas, que se detallarán más adelante.

En cuanto a las bases de datos, se utilizará dos PostgreSQL de distintas versiones, una 8.4 instalada en un Oracle Linux 9, montado en una máquina virtual, y una 9.4 instalada en un Windows 10 ,el host anfitrión.

El método de acceso al Oracle Linux será remoto, mediante MobaXterm, un cliente ssh para Windows 10. Para conseguir esto, se tendrá que configurar la red interna de la máquina virtual para que se permita el acceso (Anexo 1).

Una vez conseguida la replicación se efectuarán una serie de pruebas para comprobar la funcionalidad de la aplicación.

1-Estado del Arte

De cara a la creación y mantenimiento de sistemas de bases de datos, la replicación es un servicio casi imprescindible para garantizar una alta disponibilidad, rendimiento y fiabilidad. En una base de datos replicada, se guardan múltiples copias de los datos en múltiples nodos, pudiendo estar cada uno en distintas máquinas.

Con la replicación de los datos, la base de datos puede mantener el servicio si una de las maquinas falla.

En este escenario, podemos encontrar 3 tipos de servidores:

- Maestros: Procesan peticiones de escritura y lectura.
- Esclavos: Procesan peticiones solo de lectura.
- Stand-by: No son accesibles, salvo en caso de switchover.

La replicación de una base de datos implica:

- Consistencia de los datos: Mantener la integridad y consistencia de los datos es un aspecto vital importancia, como por ejemplo en las aplicaciones de misión crítica, que requieren una estricta consistencia de los datos modificados por transacciones.
- Tiempos de respuesta bajos durante la creación de la replica: La performance de la base de datos puede verse afectada durante el proceso de creación de la réplica, debido a razones de consistencia.
- Mantenimiento: Al duplicar los datos, se duplica el espacio ocupado, lo que implica tener que administrar la réplica.
- Alto tiempo de escritura: Las operaciones de escritura en la base de datos pueden ralentizarse durante periodos con una gran carga de modificaciones, ya que la transacción necesita transmitirse en múltiples copias.

Entre los motivos para replicar encontramos:

1. Fail over: Para seguir ofreciendo servicio en caso de fallo del nodo principal.
2. Data warehousing: Útil para pasar los datos del data warehause a los data mart.
3. Load balancing: Mediante la replicación, se puede distribuir la carga entre los nodos, para lograr un entorno más equilibrado.
4. Servidores Remotos: Para mantener la misma información en servidores separados geográficamente.

1.1-Tipos de replicación:

Una base de datos replicada, todos los cambios realizados a los datos deben ser aplicados en las múltiples copias, todas las copias tienen que ser idénticas. Tradicionalmente existen dos tipos de técnicas de replicación, síncrona (eager) y asíncrona (lazy).

En una replicación síncrona, todos los cambios son enviados a todas las copias nada más realizarse.

Por otra parte, la replicación asíncrona, maneja la propagación de los datos con un breve retraso desde la realización del cambio.

Otra característica que diferencia los tipos de replicación es quién puede realizar los cambios, será maestro-esclavo cuando los cambios tienen lugar primero en la copia primaria o maestro y después son transmitidos a las segundas copias o esclavos.

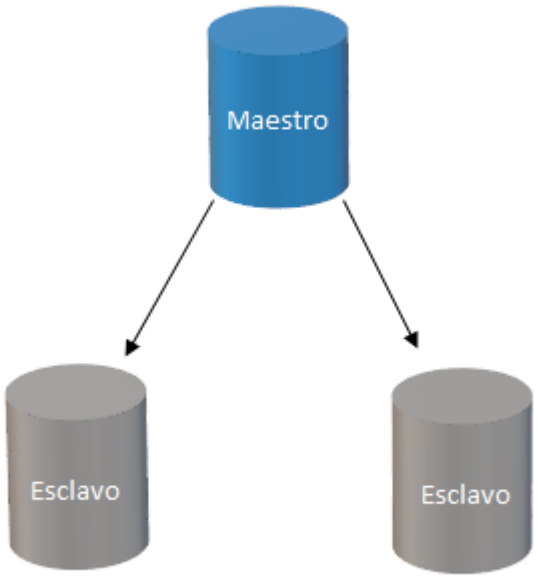

Figura 1- Estructura Maestro-Esclavo

La estructura maestro-maestro, al contrario que en la anterior, cualquier copia puede realizar cambios en los datos, dicha copia procesará el cambio y se sincronizará con las restantes.

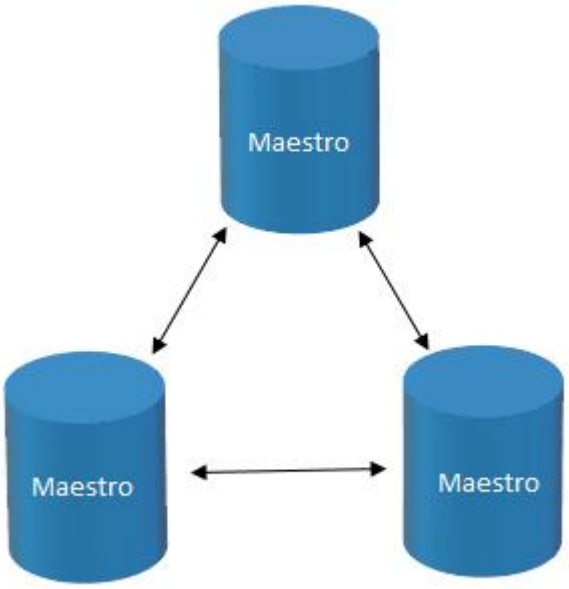

Figura 2 – Estructura Maestro-Maestro

Para garantizar la consistencia mutua en una base de datos distribuida, la transacción se procesa en la base de datos primaria, se registra en el log y se propaga a los secundarios. Un protocolo de control de atomicidad se encarga de asegurarla, o se ejecuta la transacción en todos los niveles o en ninguno. Los protocolos más conocidos son 2PC (Two phase commit) y 3PC (Three phase commit)[22].

1.2-Técnicas de replicación Síncronas

Como se ha mencionado anteriormente, existen 2 modos de estructurar los nodos según quién realiza los cambios. A continuación, se va a detallar el funcionamiento de las dos estructuras con una replicación síncrona:

Maestro esclavo Síncronos

La ejecución de la transacción empieza en la copia primaria, cuando la transacción termina, los logs generados son enviados a los secundarios, cuando la primera copia recibe la confirmación de que la transacción en las segundas copias ha finalizado correctamente, realiza el commit.

Es una técnica sencilla, ya que las modificaciones solo se pueden realizar sobre la copia primaria y no hay necesidad de coordinación. Este tipo de replicación es útil para

entornos con muchas lecturas y pocas modificaciones en los datos. Los cambios se propagan inmediatamente, disponiendo de los datos en tiempo real y asegurando la consistencia en caso de fallos.

Maestro-Maestro Síncrono

En esta técnica, mientras el ítem requerido por la transacción no esté bloqueado en todas las copias, no se puede acceder a él. Al inicio de la transacción, se envían peticiones de bloqueo a los otros participantes, hasta que el bloqueo no esté garantizado en todas las copias, la transacción continuará. Tras recibir la confirmación de bloqueo, la transacción se ejecuta en todas las copias.

La consistencia mutua de los datos está asegurada, los datos siempre van a ser idénticos en todas las copias. Una alta actividad de escritura puede provocar un exceso de bloqueos, y consecuentemente a un bajo rendimiento.

Esta técnica es útil para entornos que requieran que los datos se actualicen continuamente y en sistemas de misión crítica. Los beneficios se contrarrestan con la mayor necesidad de recursos de hardware y de red.

1.3-Técnicas de replicación asíncronas

La replicación asíncrona evita los costes de la replicación síncrona, gracias a que se proporciona una respuesta al cliente antes de la coordinación entre copias. A continuación, se detalla el funcionamiento, en modo asíncrono, de las dos estructuras de replicación de bases de datos:

Maestro-Esclavo asíncrono

Las transacciones de actualización se transmiten de la copia primaria a las segundas copias y se ejecutan en las mismas. Normalmente, los logs se propagan entre las segundas copias con un retraso después de haber hecho commit en la primaria.

Las transacciones de lectura son ejecutas en las segundas copias En cada segunda copia, se almacenan las actualizaciones en una cola FIFO. Un proceso de refresco se encargará de asegurar la concurrencia entre las transacciones de solo lectura y las de actualización transmitidas por la copia primaria.

La replicación asíncrona maestro-esclavo, es útil cuando hay muchas lecturas y pocas escrituras. La propagación de los datos es en diferido, por lo que las modificaciones se toman un tiempo en realizarse desde el commit en la copia primaria. También existe la posibilidad que, al realizar una lectura local, esta no incluya las modificaciones más recientes realizadas en la copia primaria.

Maestro-Maestro asíncrono

La transacción se ejecuta en el maestro delegado, el cliente obtiene una respuesta, transcurrido un tiempo desde el commit, las modificaciones se propagan al resto de

segundas copias. Como en el otro case de replicación asíncrona, se almacenan las actualizaciones en una cola FIFO y un proceso de refresco se encarga de su realización.

La propagación de las modificaciones, al ser en diferido puede causar inconsistencia de los datos, pudiendo incluso perder modificaciones en caso de caída.

1.4-Maestro-Maestro vs Maestro-Esclavo

La replicación multimaestro es una solución más elegante, ya que a diferencia de en la replicación Maestro -Esclavo, no necesita distribuir la carga entre las copias, como los datos son replicados en todos los nodos, las modificaciones tiene que ser ejecutadas en todos los nodos por igual. Una manera de mejorar el rendimiento de la replicación multimaestro, es procesando las operaciones solo en un sitio y mandando los resultados al resto de nodos, con esto, la transmisión de los datos entre las copias sería más robusta ante posibles fallos.

1.5-Asincrono(Lazy) vs Síncrono(Eager)

En la replicación Síncrona, se modifican todas las réplicas como parte de una sola transacción, también es conocida como replicación pesimista, ya que utiliza bloqueos y esperas hasta que la transacción finaliza.

Todo lo contrario, ocurre en la replicación Asíncrona, la transacción se transmite de réplica en réplica sin incurrir en bloqueos entre ellas. Los conflictos deben ser detectados y resueltos, ya sea por parte de la aplicación replicadora o el administrador de la base de datos.

Aunque la propagación de las modificaciones se realiza con un breve retraso, la replicación asíncrona tiene un tiempo de respuesta más rápido que la síncrona, esto se debe a las posibles esperas y deadlocks de la replicación síncrona.

En cuanto a la consistencia de los datos, solo la replicación síncrona lo garantiza, al esperar la confirmación de todos los nodos antes de hacer commit. La replicación asíncrona no lo garantiza, al realizar commit en la copia primaria sin comprobar el estado de las segundas copias.

2-Replicación en PostgreSQL

PostgreSQL es un sistema de base de datos objeto-relacional de código abierto creado por el departamento de Ciencias de la computación de Berkeley de la Universidad de California y con más de 15 años de desarrollo activo. Su arquitectura posee una gran reputación gracias a su fiabilidad, integridad de los datos y buen funcionamiento.

Puede instalarse en un gran número de sistemas operativos, incluyendo Linux, Unix y Windows.

Entre las soluciones de replicación más conocidas, se encuentran:

1. Almacenamiento compartido
2. Replicación en streaming
3. Slony
4. Londiste
5. BDR
6. Bucardo
7. Mammoth
8. Rubyrep

De las herramientas anteriores, solo la replicación en streaming es nativa de PostgreSQL, el resto de ellas son módulos de código abierto desarrollados por terceros.

2.1-Almacenamiento compartido

Mediante esta opción, se evitan costes de sincronización, ya que solo existe una copia de la base de datos. Funciona mediante un array de discos compartidos por varios servidores, si la base de datos principal falla, el servidor standby reacciona y monta y arranca la base de datos automáticamente, esto permite un rápido failover sin pérdida de datos.

Esta es una solución muy común en el almacenamiento en red. La desventaja que tiene este método se manifiesta si el fallo se origina en el disco compartido, convirtiendo tanto la copia primaria como la secundaria en inservibles. Otra pega, es que mientras el servidor primario esté en funcionamiento, el servidor standby no podrá acceder a los datos.

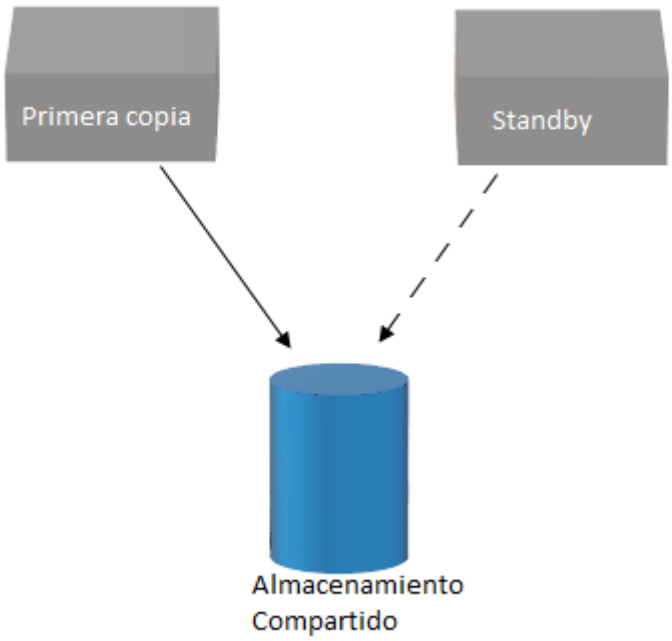

Figura 3 – Servidores con almacenamiento compartido

2.2-Replicación Streaming

La Replicación Streaming es el sistema nativo de PostgreSQL de replicación, este tipo de replicación solo puede ser de estructura Maestro-Esclavo.

En La replicación Streaming, la copia standby se conecta con la copia primaria, la cual envía sus registros WAL a la standby según los genera. Las copias standby eliminan periódicamente los WAL recibidos, para evitar una carga excesiva del disco.

La replicación streaming de PostgreSQL es asíncrona por defecto. Si el maestro falla, algunas de las transacciones realizadas, puede que no se hayan replicado a la copia standby, provocando una pérdida de datos. La cantidad de datos perdidos será proporcional al tiempo de retraso de la replicación durante el fallo.

En el modo de replicación asíncrono, cada commit de una transacción de escritura esperará hasta recibir la confirmación de que la transacción ha realizado commit en todas las copias, salvo en las transacciones pesadas, como cargas masivas de datos o construcción de índices.

Para cambiar a modo síncrono, en el archivo *postgresql.conf* del servidor maestro ,el parámetro:

- *syncronous_standby_names*: Especifica la lista de servidores standby , no debe estar vacio.
- *syncronuos_commit*: Fijándolo en *on* .
- *wal_level:* Debe ser cambiado a *hot_standby.*

En el esclavo o standby, también en el archivo postgresql.conf , hay que configurar una serie de parámetros:

- *Hot_standby*: El valor de este parámetro puede ser on u off, y especifica si el servidor tiene la habilidad de ejecutar transacciones de solo lectura mientras se encuentre en modo standby. Debe fijarse en on.

A parte de esto, habría que crear un archivo *recovery.conf* con los parámetros:

- *standby_mode* ='on'
- *primary_conninfo= String de conexión con el maestro*
- *trigger_file* = Ruta del archivo de los disparadores que manejan el fin de la replicación.
- *restore_command* = Especifica el comando de recuperación.

Para configurar la conexión entre maestro y esclavo, hay que especificar las direcciones IP por las que el maestro aceptará conexiones. En el archivo *potgresql.conf* , del maestro habrá que configurar los parámetros *Listen_adresse*s y *port.* Por otra parte, en el archivo *pg_hba.conf*, controla la autenticación de los clientes con el servidor, aquí se especifica a que bases de datos se va a conectar, con que usuario, su IP y el método de autenticación

```
# TYPE   DATABASE        USER            ADDRESS                 METHOD
host     postgres        all             192.168.12.10/32        md
```

2.3-Slony

Slony es un módulo de replicación de transacciones Maestro-Esclavo asíncrono instalado por defecto en PostgreSQL, su funcionamiento se basa en los triggers. Su nombre proviene del ruso, significa elefante, un claro guiño al logo de PostgreSLQ.

Entre sus funcionalidades, destaca:

- Puede replicar entre dos bases de datos PostgreSQL de distintas versiones.
- Puede replicar datos entre dos sistemas operativos distintos.
- Permite qué tablas replicar del maestro al esclavo.
- Permite replicar unas tablas a un esclavo y el resto a otro esclavo.
- Permite que distintos servidores de base de datos sean maestros de distintas tablas.
- La replicación se produce en cascada, un esclavo puede recibir datos replicados desde otro esclavo, reduciendo así la carga en el maestro.
- Permite realizar un switchover controlado.
- Soporta failover en caso de fallo del nodo maestro.

Su última versión, 2.2.6, requiere una versión de PostgreSQL 8.3 o superior, para versiones inferiores solo Slony 1.2 es compatible.

En los sistemas Linux, en cada nodo que participa en el cluster (agrupación de bases de datos), correrá un proceso llamado slon, que se encargará de manejar la actividad replicadora para ese nodo. En cambio, en los sistemas Windows solo se ejecuta un slon por host.

Slony también tiene un procesador de comandos propios, slonik, que procesa scripts de configuración del cluster con slony.

Limitaciones de Slony:

- No admite cambios en archivos BLOB.
- No admite cambios DDL mediante comandos.
- No admite cambios en usuarios y roles.
- Requiere modificaciones en el núcleo de PosgreSQL.

La razón principal de estas limitaciones es, que los triggers de PostgreSQL no soportan los cambios en el schema ni en objetos grandes (BLOBS) .

También necesita que se editen los archivos pg_hba. conf y el postgresql.conf para configurar la conexión del maestro con los esclavos en el cluster.

Los pasos a seguir para configurarlo son:

1. Crear el cluster en el nodo maestro.
2. Añadir esclavos al cluster.
3. Configurar las rutas a los nodos en cada uno de ellos.
4. Modificar los archivos pg_hba. conf y el postgresql.conf.
5. Crear un set de replicación y añadir las tablas que se quieren replicar al set.
6. Suscribir al set a los nodos esclavos.

2.4-Londiste

Londiste es un sistema de replicación maestro-esclavo asíncrono que forma parte del paquete de herramientas skyTools de Skype (Basadas en Python) para PostgreSQL. Londiste está basado en triggers, y permite replicar a elección un conjunto de tablas del maestro a los esclavos

Conceptos básicos para la utilización de Londiste:

- El maestro es llamado proveedor, y los esclavos son sus suscriptores.
- En el proveedor, se ejecuta un Ticker, que dictamina la frecuencia de los eventos de la replicación.
- En cada suscriptor, existirá un demonio encargado de la ejecución de la replicación.
-

➢ **Ventajas:**

1. Fácil de configurar y administrar.

➢ **Desventajas:**

1. No soporta la ejecución de scripts SQL.
2. No soporta la replicación en cascada.
3. Escasa documentación.
4. Delicado con los cambios DDL.
5. Permite realizar cambios en el esclavo.
6. Requiere modificaciones en el núcleo de PosgreSQL.

2.5-BDR

BDR(Bi-Directional-Replication) es un sistema de replicación asíncrono multimaestro de código libre para PostgreSQL, especializado en bases de datos distribuidas geográficamente desarrollado por 2ndQuadrant , una consultoría especializada en PostgreSQL.

BDR no depende del uso de disparadores para la detección de modificaciones y la posterior replicación de estas. Utiliza el WAL, utilizando un mecanismo de extracción de cambios desarrollado.

➢ **Ventajas:**

1. Soporta hasta 48 nodos.
2. Bajo consumo de recursos.

3. No necesita un demonio en cada nodo, BDR ejecuta sus procesos en el mismo PostgresQL.

- **Desventajas:**
  1. Solo soporta versiones de PostgreSQL anteriores a la 9.4.
  2. No es compatible con Windows.
  3. Es un sistema poco maduro.
  4. Requiere modificaciones en el núcleo de PosgreSQL.

2.6-Bucardo

Sistema de código libre de replicación asíncrona basada en triggers, que admite tanto la replicación multimaestro como la maestro-esclavo asíncrono para PosgreSQL desarrollado por Jon Jensen y Greg Sabino Mullane.

Bucardo se instala en una sola base de datos, generalmente en el maestro, en la que se indica la ubicación del resto de nodos. En cada nodo se ejecutará un demonio en Perl, que se encargará de la replicación entre todos los nodos.

Permite el balanceo de carga y data warehousing en los esclavos y permite la escritura en los nodos esclavos.

Bucardo funciona entre distintos sistemas operativos, pero el maestro, donde debe estar instalado, tiene que ser una distribución de Linux.

- **Ventajas:**
  1. En el modo multimaestro, proporciona resolución de conflictos.
  2. La propagación es en cascada.
  3. El maestro debe ser PostgreSQL, pero el objetivo puede ser PostgreSQL, MySQL, Redis, Oracle, MariaDB, SQLite, o MongoDB.
  4. Permite la replicación desde versiones PostgreSQL 8.1 a PostgreSQL 9.6.
  5. No requiere modificaciones en el núcleo de PosgreSQL.

- **Desventajas:**
  1. No replica cambios DDL .
  2. No soporta archivos BLOB.
  3. Puede ser un poco difícil de instalar debido a varias configuraciones y dependencias necesarias a nivel del Sistema Operativo.

Al ser la herramienta elegida para el proyecto, se detallará a fondo su funcionamiento más adelante.

2.7-Otros sistemas

Otras opciones ya fuera de mantenimiento son:

- **Mammoth**: Este proyecto no recibe actualizaciones desde 2010. Proporciona replicación asíncrona maestro-esclavo, con la particularidad de soportar archivos BLOB. La última versión estable es la 1.8 para PostgreSQL 8.3.

- **Rubyrep**: Al llevar más de 2 años sin actualizaciones, parece que el proyecto se encuentra estancado. Es una herramienta asíncrona de maestro-maestro.

Herramienta	Método de Replicación	Modo de sincronización
Streaming Replication	Maestro-Esclavo	Síncrono/Asíncrono
Slony	Maestro-Esclavo	Asíncrono
Londiste	Maestro-Esclavo	Asíncrono
Bucardo	Maestro-Esclavo/Maestro-Maestro	Asíncrono
BDR	Maestro-Esclavo/ Maestro-Maestro	Asíncrono
Mammoth	Maestro-Esclavo	Asíncrono
RubyRep	Maestro-Maestro	Asíncrono

Tabla 1 -T. Resumen de las herramientas estudiadas

3-Descripción de necesidades

Para este proyecto, se quiere replicar una base de datos PostgreSQL 8.4 alojada en una máquina virtual Oracle Linux 6.9. El otro nodo, será una base de datos PostgreSQL 9.4, alojada en un ordenador portátil con Windows 10 por sistema operativo.

La estructura de la replicación debe ser Maestro-Maestro, para poder realizar operaciones de escritura en ambas bases de datos y que se transmitan los cambios realizados bidireccionalmente.

En cuanto a la sincronización, deberá ser asíncrona , ya que para PostgreSQL, de momento no existe ningún modulo que soporte la replicación síncrona Maestro-Maestro.

4-Elección de sistema replicador

Para la realización de este proyecto, como indica su título, el tipo de replicación buscada es maestro- maestro, solo 2 de las herramientas anteriores pueden implementar esta opción, BDR y Bucardo.

De acuerdo con lo expuesto en el punto anterior, al requerir que la replicación se realice entre dos versiones de PostgreSQL muy distantes (8.4 y 9.4), se descarta la opción de BDR, ya que no funciona en versiones anteriores a la 9.4.

Por lo tanto, se instalará Bucardo en el Oracle Linux 6.9 alojada en la máquina virtual y se configurará para que replique con la base de datos instalada en Windows 10.

Bucardo solo ofrece una sincronización asíncrona, esto no supone ningún inconveniente, todo lo contrario, con los recursos disponibles, es la mejor opción, una sincronización síncrona exigiría un consumo de recursos mayor, que el hardware utilizado podría no soportar.

5-Instalación PostgreSQL 8.4

La instalación de Bucardo debe realizarse en una distribución de Linux, el sistema operativo elegido ha sido Oracle Linux 6.9, ya que viene con los paquetes de PostgreSQL 8.4 en la configuración.

Antes de instalar Bucardo, la base de datos PostgreSQL 8.4 tiene que estar instalada:

1. El primer paso es crear el directorio donde se ubicarán los datos de Postgres, el directorio tiene que crearse como root, y una vez creado, conceder permisos al usuario, en este caso, al usuario 'produccion'.

```
[root@localhost ~]# mkdir /postgres
[root@localhost ~]# chown produccion:produccion /postgres
```

Imagen 1- Creación del directorio Postgres

2. Después hay que añadir la variable PGDATA en el bash_profile, aquí se especificará la ruta del directorio creado en el paso anterior, donde irán los datos de PostgreSQL.

```
[produccion@localhost postgres]$ vi .bash_profile
```

Imagen 2 – Apertura del bash profile

Siendo este el resultado del bash_profile:

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH

PGDATA=/postgres
export PGDATA
```

Imagen 3 – Edición del bash profile

Una vez modificado el bash_profile, se requiere reiniciar la máquina para que los cambios realizados surjan efecto. Una vez reiniciado, se ejecuta echo a la variable, para comprobar que apunta a la ruta correcta:

```
[produccion@localhost ~]$ echo $PGDATA
/postgres
```

Imagen 4 – Comprobación de la variable PGDATA

3. Con la variable PGDATA definida, el siguiente paso es ejecutar *initdb ,* una vez ejecutado, si se ha completado sin errores, debe dar una salida como la siguiente:

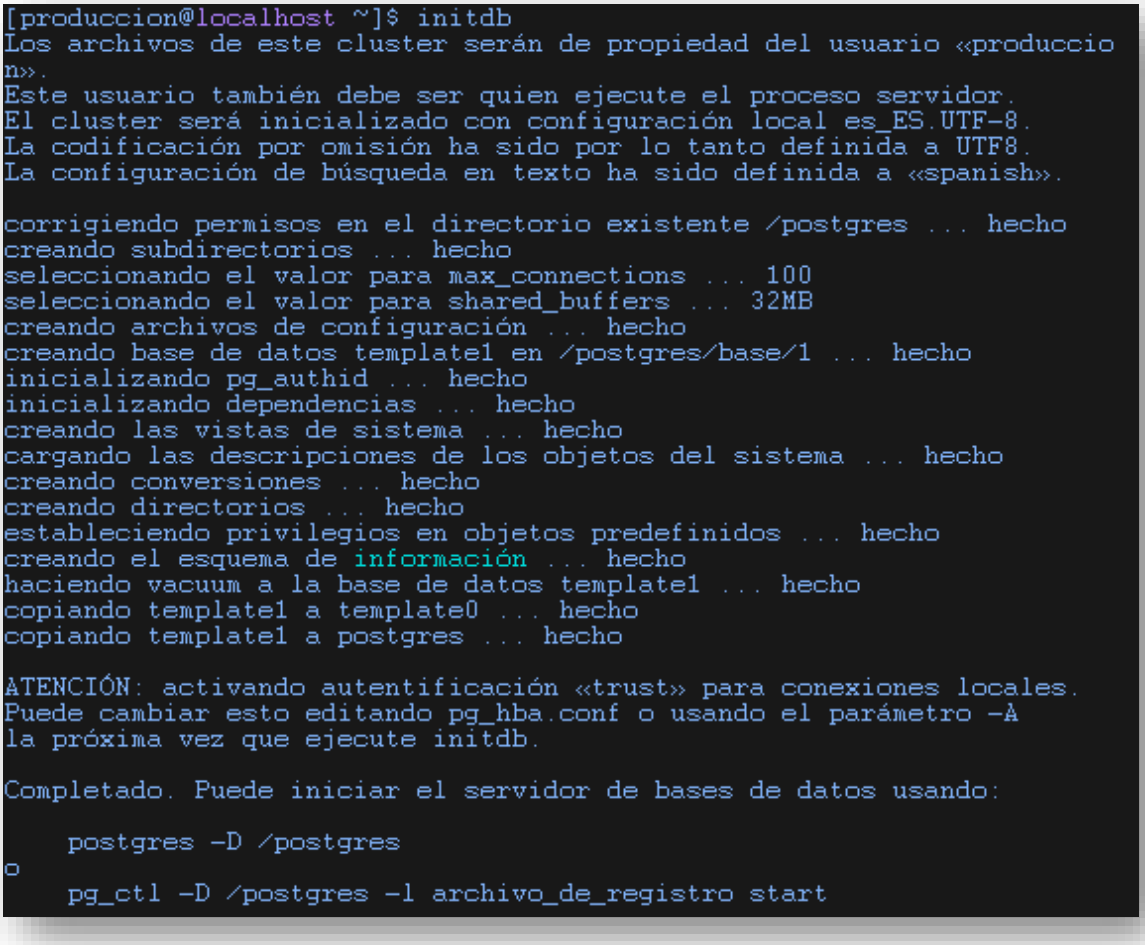

Imagen 5 – Resultado de la ejecución del comando initdb

4. El último paso para tener PostgreSQL corriendo es ejecutar pg_ctl start:

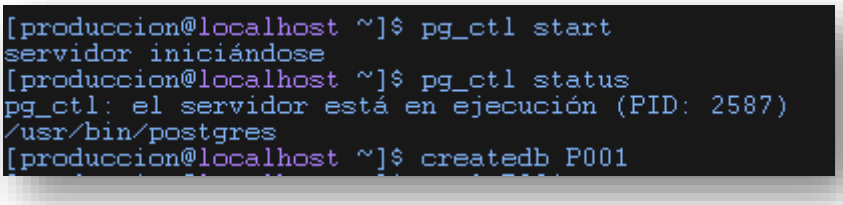

Imagen 6 – Inicio del servidor

- Pg_ctl start: Arranca el servidor PostgreSQL.
- Pg_ctl status: Comprueba el estado del servidor.
- Createdb <<nombre>>: Crea una base de datos.

Con esto ya está instalado PostgreSQL 8.4 y con una base de datos corriendo, lista para instalar Bucardo y empezar con la replicación.

6-Instalación de Bucardo

La versión de Bucardo que se va a instalar es la 5.4.1, la instalación va a realizarse vía paquete , descargando Bucardo.tar.gz desde la página oficial del programa.

Pasos a seguir:

1. Descomprimir el paquete dbix_safe.tar.gz. Mediante la instalación de este paquete, se cumple el prerrequisito de tener instalado el módulo safe de perl. Este paquete se puede encontrar en la página oficial de Bucardo.

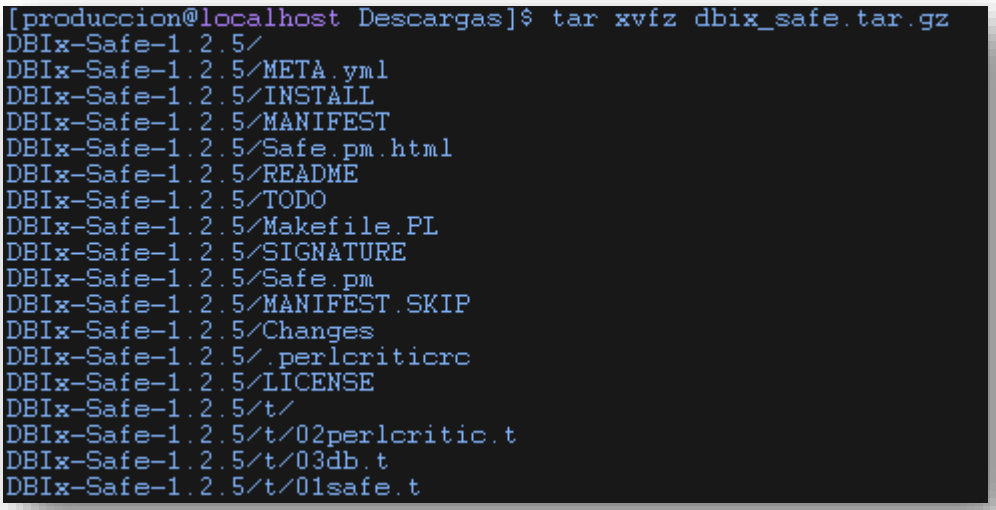

Imagen 7 – Descompresión del Paquete DBIX

Una vez se ha descomprimido el paquete, hay que posicionarse en el directorio DBIx-Safe-1.2.5, para crear e instalar los archivos perl:

```
[root@localhost DBIx-Safe-1.2.5]# perl Makefile.PL
Writing Makefile for DBIx::Safe
[root@localhost DBIx-Safe-1.2.5]# make
cp Safe.pm blib/lib/DBIx/Safe.pm
Manifying blib/man3/DBIx::Safe.3pm
Created Safe.pm.html
[root@localhost DBIx-Safe-1.2.5]# make test
[root@localhost DBIx-Safe-1.2.5]# make install
Created Safe.pm.html
Installing /usr/local/share/perl5/DBIx/Safe.pm
Installing /usr/local/share/man/man3/DBIx::Safe.3pm
Appending installation info to /usr/lib64/perl5/perllocal.pod
```

Imagen 8 – Creación e instalación de archivos Perl

Después de ejecutar estos comandos, el módulo safe de perl debería estar correctamente instalado.

Durante este paso, puede que surja algún problema si el sistema no tiene instalado algunos de los prerrequisitos de perl. En este caso ocurrió lo siguiente:

```
[produccion@localhost DBIx-Safe-1.2.5]$ perl Makefile.PL
Checking if your kit is complete...
Looks good
Warning: prerequisite DBD::Pg 1.49 not found.
```

Imagen 9 – Prerrequisitos de Perl no encontrado

El workaround para este problema ha sido reinstalar el módulo plperl:

```
[root@localhost ~]# yum install postgresql-plperl
```

Imagen 10 – Instalación módulo Plperl

2. El segundo paso consiste en descomprimir el paquete *Bucardo-5.4.1.tar.gz*, y ejecutar los comandos perl Makefile.PL y make install, dentro del directorio descomprimido *Bucardo-5.4.1:*

```
Bucardo-5.4.1]# perl Makefile.PL
Bucardo-5.4.1]# make install
```

Imagen 11 – Instalación Perl para Bucardo

3. Para finalizar la instalación de Bucardo, hay que crear el directorio Bucardo en la ruta /var/run , donde se almacenará el Id del proceso de Bucardo y ejecutar Bucardo install:

```
[root@localhost Bucardo-5.4.1]# mkdir /var/run/bucardo
[root@localhost Bucardo-5.4.1]# bucardo install
```

Imagen 12 – Instalación de Bucardo

Al ejecutar el comando Bucardo install, en pantalla se muestra el siguiente menú de configuración:

```
This will install the bucardo database into an existing Postgres clust
er.
Postgres must have been compiled with Perl support,
and you must connect as a superuser

Current connection settings:
1. Host:           <none>
2. Port:           5432
3. User:           postgres
4. Database:       bucardo
5. PID directory:  /var/run/bucardo
Enter a number to change it, P to proceed, or Q to quit: 3
```

Imagen 13 – Configuración de la instalación de Bucardo

En este menú, se configura Bucardo de acuerdo con la base de datos creada anteriormente, la opción Host y Puerto coinciden en este caso, en User, en vez de postgres, será producción, y en Database, el nombre de la base de datos creada, P001:

```
Current connection settings:
1. Host:           <none>
2. Port:           5432
3. User:           produccion
4. Database:       P001
5. PID directory:  /var/run/bucardo
Enter a number to change it, P to proceed, or Q to quit: P

Postgres version is: 8.4
Creating superuser 'bucardo'
Attempting to create and populate the bucardo database and schema
Database creation is complete

Updated configuration setting "piddir"
Installation is now complete.
```

Imagen 14 – Configuración de la instalación de Bucardo II

7-Configuración de la replicación

Para poner en marcha la replicación, primero hay que indicar a PostgreSQL qué bases de datos se pueden conectar, su IP, y el puerto en el que conectarse. Para ello hay que modificar los archivos pg_hba.conf y postgresql.conf de una de las bases de datos, ya que solo es necesario que una abra el canal de conexión:

- En la máquina Windows 10:
  - En postgresql.conf:
    - Listen_adresses= '*': Con esto le indicamos que escuche las peticiones de todas las ip que entren.
    - Port=5432: Es el puerto por el que escucha por defecto.
  - En pg_hba.conf se añade la siguiente línea :

    - host all all 127.0.0.1/32 md5

    En esta línea, le estamos indicando que las IP dentro de nuestra red se conecten mediante autenticación md5 con cualquier usuario a cualquiera de las bases de datos.

Una vez asegurada la conexión entre los dos nodos, y teniendo ya instalado Bucardo, hay que indicarle a este último donde se ubican las tablas a replicar.

Antes de empezar a replicar, conviene conocer los términos que utiliza Bucardo en el proceso:

- *Database*: Aquí se definen las bases de datos que van a participar en la replicación.
- Table: Bucardo permite elegir qué tablas se desean replicar, en el caso de querer replicar todas, le debe seguir all como argumento.
- *Herd*: Es el conjunto de tablas que van a participar en la replicación, es útil en la replicación maestro-esclavo, para organizar las tablas que se quieren replicar con origen en el maestro o en el esclavo.
- *Sync*: Aquí se especifica el tipo de sincronización que va a usarse en durante la replicación. Hay 3 opciones:
  - *Fullcopy*: Borrará y reescribirá por completo la tabla destino.
  - *Pushdelta*: Actualizará los cambios realizados en el maestro al esclavo.
  - *Swap*: Escribe los cambios realizados en todos los nodos, de manera bidireccional.
- *Relgroup*: Es el conjunto de tablas y relaciones que se van a replicar.
- *Dbgroup*: Conjunto de base de datos a replicar.

El primer paso, es añadir a Bucardo las bases de datos, tanto la local como la remota, mediante el comando: ***Bucardo add db <nombre de la base de datos en Bucardo> dbname=<nombre de la base de datos a replicar>:***

```
[produccion@localhost ~]$ bucardo add db P1 dbname=P001
Added database "P1"
[produccion@localhost ~]$ bucardo add db P2 dbname=P002  host=10.0.2.2 pass=oracle
Added database "P2"
```

Imagen 15 – Añadir una base de datos a Bucardo

Para añadir una base de datos local, solo es necesario indicarle el nombre, para una remota, como se puede observar en la imagen de arriba, hay que "decirle" a Bucardo donde está el servidor y su contraseña, para que pueda conectarse.

Para comprobar que ambas bases de datos se han añadido sin problemas, se puede ejecutar el comando Bucardo *list db*, que listará las bases de datos añadidas:

```
[produccion@localhost ~]$ bucardo list db
Database: P1  Status: active  Conn: psql -U bucardo -d P001
Database: P2  Status: active  Conn: psql -U bucardo -d P002 -h 10.0.2.2
```

Imagen 16 – Listar las bases de datos añadidas en Bucardo

El siguiente paso es añadir las tablas que se quieran replicar:

```
[produccion@localhost ~]$ bucardo add  table persona relgroup=relgroup1  db=P1
```

Imagen 17 – Añadir una tabla a Bucardo

En la sentencia anterior, le estamos diciendo a Bucardo que añada la tabla persona, de la base de datos P1(nombre dado a la base de datos P001 dentro de Bucardo) y que la incluya en el relgroup1.

Ahora solo falta crear la sincronización entre las dos bases de datos, para ello, primero se va a crear un *dbgroup*, con las dos bases de datos a replicar:

```
[produccion@localhost ~]$ bucardo add dbgroup group1  P1:source P2:source
```

Imagen 18 – Crear un grupo de bases de datos en Bucardo

Con esta sentencia, se ha creado el *group 1*, que contiene las bases de datos referidas como P1 y P2, ambas definidas como *source*, lo que está indicando a Bucardo que la replicación será bidireccional, es decir Maestro-Maestro. Si se quiere que la replicación sea Maestro-Esclavo, o simplemente añadir un esclavo a la replicación, habría que cambiar el parámetro a *target*, que indica que la base de datos solo será objetivo.

Ya solo falta un paso para terminar con la configuración de la replicación, añadir la sync, mediante add sync, se creará una nueva sincronización entre las bases de datos definidas:

```
bucardo add sync sync1 relgroup=relgroup1 dbs=group1 onetimecopy=2 conflict_strategy=bucardo_latest autokick=1
```

Imagen 19- Creación de la sync de Bucardo

- En *relgroup* se definen las tablas y relaciones a replicar.
- En *dbs* se especifican las bases de datos que se van a replicar, en este caso, se ha añadido el grupo creado anteriormente (P1 y P2).
- El parámetro *onetimecopy* indica si se va a realizar una copia de los datos entre los nodos, admite los siguientes valores:
  - 0: Desactivado.
  - 1: Activado.
  - 2: Activado solo si un nodo contiene datos y el otro no.
- En *conflict_strategy* se determina la estrategia a seguir en caso de conflictos, admite los siguientes valores:
  - *Bucardo_source*: Las filas del nodo origen siempre ganan.
  - *Bucardo_target*: Las filas del nodo objetivo siempre ganan.
  - *Bucardo_skip*: En caso de conflicto, se aborta la replicación del objeto.
  - *Bucardo_random*: El nodo ganados se elige al azar.
  - *Bucardo_latest*: La fila que fue cambiada más recientemente gana. Es la estrategia por defecto.
  - *Bucardo_abort:* En caso de conflicto, la sincronización se aborta.

Con esto, ya solo hace falta ejecutar *Bucardo start* para empezar a replicar y *Bucardo status* para comprobar que la replicación funciona sin problemas.

```
 Name    State    Last good    Time    Last I/D    Last bad    Time
=======+========+============+=======+===========+===========+=======
 sync1 | Good   | 23:13:16   | 54s   | 0/0       | none      |
```

Imagen 20 – Estado de la sync de Bucardo

8-Problemas comunes y sus soluciones

En caso de haber algún problema, será en este punto cuando surjan, los errores más comunes se dan con los módulos de Perl, que o no están instalados o están en una ruta incorrecta.

Entre ellos, los paquetes que dan más problemas son 2, *Boolean.pm* y *DBD::Pg.* . La solución para este tipo de problemas es reinstalar los paquetes, fácil a priori, pero a la hora de instalarlos, no se encuentran en ningún repositorio.

La solución es instalar el módulo CPAN de Perl, que contiene todo los paquetes y módulos relacionados con Perl. Se invoca mediante el comando *cpan <paquete a instalar>*.

Otro tipo de problemas comunes son los ocasionados por falta de permisos:

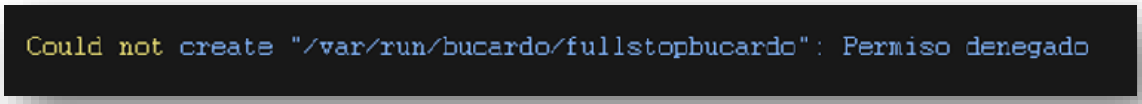

Imagen 21 – Problema de permisos

La solución es simple, dar permisos al usuario para que pueda acceder al directorio y al archivo, mediante el comando de Linux *chown*.

El ultimo tipo de problema que puede surgir, es la falta de directorios y/o archivos, como, por ejemplo:

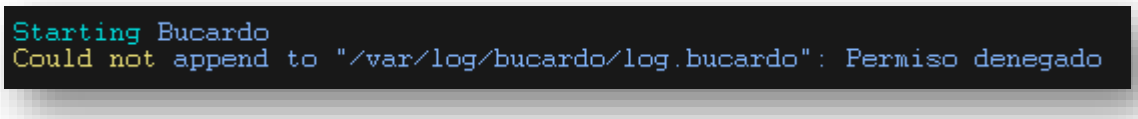

Imagen 22 – Problema de permisos II

En este caso, hay que iniciar sesión como *root*, crear el directorio Bucardo en la ruta especificada con el comando *mkdir*, y dentro de este crear el archivo log.*Bucardo* con *vi*.

9-Pruebas

En este apartado, se van a realizar pruebas sobre el entorno para probar la funcionalidad de la replicación.

1- Actuación frente a una parada inesperada:

   Para realizar estas prueba, se va a cambiar el parámetro autokick a 0 de la sync, con lo que la replicación no será automática, sino controlada por el usuario, para así tener tiempo de realizarla.
   Para simular una parada del servidor, se ejecutará el comando pg_ctl stop -m i, que forzará el apagado de PostgreSQL. Antes de apagarlo, se ha realizado un *update* en dicho servidor, sin antes activar la replicación.

   Tras reiniciar el servidor, mediante la sentencia *Bucardo kick sync1*, arranca la replicación. Al arrancar, Bucardo, analiza las transacciones en la cola de salida
   y la envía al otro nodo las modificaciones realizas antes de la caída.

   Si la caída se produce en el nodo en el que no se realiza el *update*, al volver a arrancar el nodo, Bucardo sincronizará automáticamente los dos nodos, asegurándose de que ambos tengan los mismos datos.

2- Añadir una columna a la tabla

   Al añadir una nueva columna a la tabla, al no reconocer los cambios DDL, Bucardo no hará nada que afecte a la columna.
   Para que funcione, será necesario crear manualmente la nueva columna en el otro nodo y ejecutar el siguiente comando *Bucardo update table <nombre>*, para que Bucardo reconozca la nueva columna.

3- Añadir una tabla a la sincronización

   Si se quiere añadir una nueva tabla a la sincronización, antes de nada, esta debe estar presente en todas las bases de datos, ya que Bucardo no replica cambios DDL.
   Una vez creada la tabla en todas las bases de datos, hay que indicarle a Bucardo donde está la nueva tabla, mediante:

```
[produccion@localhost /]$ bucardo add table mascotas relgroup=relgroup1 db=P1
```

Imagen 23 – Añadir una tabla a la sync de Bucardo

Con esa sentencia, le estamos indicando a Bucardo que incluya la tabla 'mascotas' en el relgroup relgroup1 (conjunto de tablas que se replican). Tras esto, hay que ejecutar Bucardo reload, para refrescar la configuración de Bucardo y se añada la nueva tabla a la replicación.

4- Actuación frente a conflictos

Para realizar esta prueba, se ejecuta el siguiente update:

```
-psql P001 -c "update persona set direccion='Alcla' where nombre='Adrian'" &&  psql -h 10.0.2.2 -U postgres -d P002 -c "update persona set direccion='Atocha' where nombre='Adrian'"
```

Ambos update se ejecutan simultáneamente, cada uno en un nodo, sobre el mismo campo de la misma tabla, causando un conflicto.

La estrategia por defecto de Bucardo ante conflictos es el Bucardo_lastest, que ante un conflicto, elige la última en llegar.

Por lo tanto, el update que se ejecuta es el último, cambiando el campo dirección a Atocha.

5- Añadir otra base de datos a la replicación

El primer paso es crear la base de datos, con la misma estructura DDL que las otras dos. Una vez creada, hay que añadirla a Bucardo:

```
[produccion@localhost /]$ bucardo add db P3 dbname=P003 host=10.0.2.2
Added database "P3"
[produccion@localhost /]$ bucardo update dbgroup group1 P1:source  P2:source
P3:source
Added database "P3" to dbgroup "group1" as source
```

Imagen 24 – Añadir base de datos a la sync de Bucardo

Tras añadirla a Bucardo, para que forme parte de las bases de datos replicadas, tiene que pertenecer al mismo dbgroup que las otras bases de datos. Una vez ya incluida en el dbgroup, hay que asegurarse de que la nueva base de datos añadida dispone de los triggers de Bucardo necesarios para la replicación, para esto, se ejecuta el comando *Bucardo validate*:

```
[produccion@localhost /]$ bucardo validate sync1
Validating sync sync1 ... OK
```

Imagen 25 – Validación de la sync de Bucardo

10-Conclusiones

Para comenzar con el análisis y conclusiones sobre el trabajo realizado, se empezará por los problemas y dificultades que han surgido a lo largo de su elaboración. El primer problema en el camino ha sido la falta de conocimientos sobre el tema. Han hecho falta muchas horas de documentación e investigación para lograr una base sólida para empezar con el proyecto.
El segundo gran problema, ha sido el conseguir una configuración óptima de la máquina virtual y del sistema operativo para que fuese compatible con el software elegido.

Después de conseguir la configuración adecuada, la instalación de tanto PostgreSQL como Bucardo, ha sido un proceso más rápido y dinámico, aunque la documentación acerca de Bucardo no abunda en la red, es suficiente para ponerlo en marcha.

En el ámbito teórico, he aprendido acerca de los distintos métodos de replicación existentes, y las ocasiones en las que conviene usar unos y otros:

La replicación síncrona, es aconsejable para los entornos de misión crítica, donde es necesario garantizar la consistencia y disponibilidad de los datos y las segundas copias tengan los mismos datos que la primaria. El punto negativo que tiene este tipo de replicación es el coste que puede conllevar en el rendimiento de base de datos y en la resolución de conflictos.

La replicación asíncrona, no garantiza lo mismo que la síncrona, ya que, en caso de caída, no garantiza la propagación de las modificaciones realizadas en ese momento.
En cuanto al tiempo de respuesta, salvo en algunos casos, con la herramienta utilizada, dicho tiempo es mínimo, de un segundo como mucho.

En cuanto a la estructura protagonista de este proyecto (Maestro-Maestro), al estar compuesta por nodos funcionales, en caso de caída de uno de ellos, otro estará disponible inmediatamente.

En conclusión, existen muchas soluciones para replicar una base de datos, tanto de código libre como de pago, lo más importante a la hora de elegir una y su estructura, es el análisis previo e intensivo de las necesidades y características del sistema a replicar, ya que un mal análisis puede conducir a una estructura de replicación errónea y en consecuencia la pérdida de datos.

11-Pliego de Condiciones

Para la consecución del proyecto, ha sido necesario la utilización de las siguientes herramientas:

1- Herramientas Software utilizadas:

Para la construcción de este proyecto, ha sido necesario el siguiente software, el uso de programas distintos, podría alterar el resultado del mismo:

- Sistemas operativos: Los sistemas operativos utilizados han sido Windows 10 y Oracle Linux versión 6.9. Ninguno precisa de características especiales para la elaboración del proyecto.

- Sistema de virtualización: Para el uso de máquinas virtuales, el software usado ha sido Virtual Box de Oracle, la configuración usada en este programa no se garantiza que sea la idónea para este proyecto si se utiliza otro software.

- PostgreSQL 8.4 y PostgreSQL 9.4 como bases de datos, otras versiones también serían compatibles, siempre que sea superior a la 8.4.

- El programa encargado de la replicación ha sido la última versión de Bucardo.

- En la maquina en la que se instaló Bucardo es necesario que tenga la última versión del módulo de Perl.

- MobaXterm, para conectarse remotamente al servidor Oracle Linux.

2-Herramientas Hardware utilizadas:

- Ordenador portátil Asus A53S del 2011, con 6 GB de RAM, procesador Intel i7-2670 y 500GB de disco duro.

12-Presupuesto

Este proyecto ha consistido en la investigación de las distintas opciones disponibles para replicar una base de datos PostgreSQL con una estructura multimaestro. Tras la fase de investigación, se han analizado las necesidades del proyecto, y se ha implementado una de las opciones, siguiendo a esto, una fase de pruebas.

1-Mano de obra:

La consecución de este proyecto ha durado 82 jornadas, con una dedicación media diaria de 4.5h, resultando en aproximadamente 370 horas trabajadas.

Tipo de trabajo	Unidades trabajadas (horas)	Precio unitario €	Coste Total €
Investigación	250	12.00	3000.00
Técnico	120	8.00	960.00
TOTAL	370		3.960.00

2-Material intangible:

Todo el software utilizado es de código libre, por lo que no representa coste alguno, salvo el sistema operativo Windows 10 y el paquete Microsoft Office.

Producto	Precio €
Windows 10	200,00
Microsoft Office	279,00
TOTAL	479,00

3-Material:

Producto	Precio €
Portátil Asus A53S	500,00
Material de oficina	10,00
TOTAL	510,00

4-Presupuesto total:

Concepto	Coste €
Mano de obra	3.960,00
Material intangible	479,00
Material	510,00
I.V.A 21%	1.039,29
Cuota autónomos x 3 meses	792.00
TOTAL	6780.29

13-Manual de Bucardo

A continuación, se van a explicar los comandos más útiles en el manejo de Bucardo:

- **Bucardo install:**

  Instala el schema de Bucardo en un cluster de Postgres, crea el usuario y la base de datos Bucardo. Admite los siguientes parámetros:

  - --**dbuser**: Para indicar el usuario, por defecto tiene el usuario Postgres.
  - --**dbname**: Para indicar el nombre de la base de datos, el valor por defecto es Postgres.
  - --**dbport**: Para indicar el puerto por el que escucha la base de datos, por defecto tiene el puerto 5432.
  - --**pid**-dir: Para indicar el directorio PID, por defecto es /var/run/Bucardo.

- **Bucardo upgrade:**
  Actualiza una instalación de Bucardo a la última versión disponible de Bucardo.

- **Bucardo start:**
  Arranca el programa.

- **Bucardo stop**
  Fuerza la parada de los procesos de Bucardo.

- **Bucardo restart**
  Para la ejecución de Bucardo, y la reinicia una vez parada.

- **Bucardo list**
  Enumera los objetos que se le pasan como parámetro (Database,dbgroup,relgroup,sync, table, sequence y all).
  La opción all, despliega información de todos los tipos de objetos.

- **Bucardo status**
  Muestra en pantalla el estado de las sincronizaciones activas, detallando el nombre de la sync, solo mostrará los detalles de esta.

- **Bucardo add <objeto> <nombre> <parámetros>**

  Añade un nuevo objeto a Bucardo:

  - **Bucardo add db <name>**: Añade una nueva base de datos, los parámetros que soporta son:
    - **Dbname**: Nombre de la base de datos real.
    - **Dbtype**: Tipo de base de datos, Oracle, postgres, mongo…
    - **User**: El usuario con el que conectarse a la BBDD.
    - **Pass**: La contraseña para acceder a la BBDD.
    - **Host**: El host al que se va a conectar.
    - **Port**: El puerto al que conectarse.
  - **Bucardo add dbgroup <nombre> db1:source/target db2:source/target.**
  - **Bucardo add table schema.nombre** .
    - **Db**: Base de datos a la que pertenece la tabla.
  - **Bucardo add sync <nombre>**: Añade una sincronización a Bucardo.
    Admite los parámetros:
    - **Dbs**: El nombre de las BBDD que van a participar en la sincronización separado por comas.
    - **Relgroup**: Nombre del grupo de tablas y secuencias a replicar.
    - **Conflict_strategy**: La estrategia a seguir con los conflictos.
    - **Onetimecopy**: Especifica si se quiere copiar los datos actuales.
    - **Autokick**: Determina si las tablas empiezan solas la replicación al detectar cambios.

- **Bucardo update <objeto> <nombre>**

  Actualiza cualquier objeto que se le pase por el parámetro, especificando también su nombre. Soporta:

  - **Bucardo update db**: Actualiza la base de datos. Admite cambios para:
    - **Dbname.**
    - **Db.**
    - **Type.**
    - **User.**
    - **Pass.**
    - **Host.**
    - **Port.**

- **Bucardo update sync <nombre>**

  Actualiza una sincronización ya existente. Admite cambios en:

  - **Name**: Nombre de la sync.
  - **Dbs**: Grupo de bases de datos que participan en la sync.
  - **Relgroup**: Grupu de tablas y relaciones de la sync.
  - **Conflict_strategy.**
  - **Onetimecopy.**

- **Bucardo update table schema<nombre> db=<nombre>**

  Admite cambios en:

  - **Db**: La base de datos a la que pertenece la tabla.
  - **Tablename**: El nombre de la tabla.
  - **Schema**: El esquema al que pertenece.

- **Bucardo remove <objeto> <nombre>**

  Borra un objeto de Bucardo, admite los siguientes valores:

  - Db.
  - Dbgroup.
  - Relgroup.
  - Sync.
  - Table.
  - Sequence.

- **Bucardo kik <nombre sync>**

  Obliga a una sync a arrancar lo antes posible.

- **Bucardo reload config**

  Recarga la configuración de Bucardo

- **Bucardo show all**

  Muestra toda la configuración actual de Bucardo.

- **Bucardo actívate <nombre sync**

  Activa una sync.

- **Bucardo deactivate <nombre sync>**

  Desactiva una sync.

Anexo 1- Creación y configuración del entorno de trabajo

Para la realización de este proyecto, se ha utilizado el programa Oracle VirtualBox para la creación de la máquina virtual donde alojar uno de los nodos. En este anexo se detallará como se ha configurado para lograr la consecución del proyecto.
El primer paso ha sido obtener el archivo .iso del sistema operativo Oracle Linux 6.9 x64, descargado desde la página oficial de Oracle Linux.

Desde la ventana inicial de VirtualBox, hay que hacer click en el botón “Nueva”, que abre el siguiente formulario:

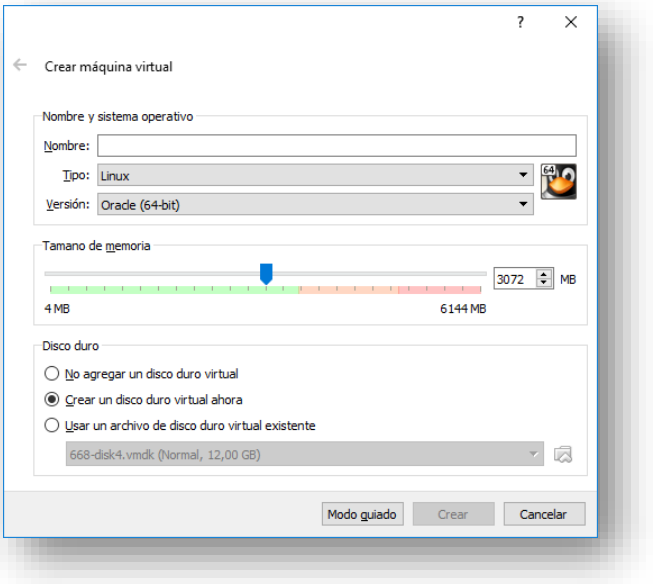

Imagen 26 – Crear máquina virtual

Se añade el nombre de máquina virtual, se selecciona el tipo y versión del sistema operativo, en este caso, Linux y Oracle, se le asigna un tamaño de memoria, según el uso que se le quiera dar y se selecciona la opción ‘crear un disco virtual ahora’.

En la siguiente pantalla hay que definir el espacio de disco duro virtual a asignar a nuestra máquina:

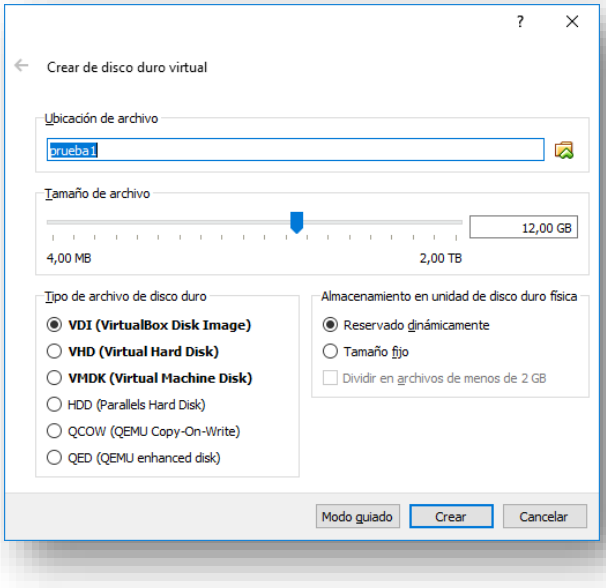

Imagen 27 – Creación disco duro virtual

Ahora hay que configurarla, haciendo click en el botón “Configurar” se accede al menú de configuración.
En la pestaña de almacenamiento, en el icono del disco, hay que seleccionar el archivo .iso para cargar el sistema operativo en la máquina virtual:

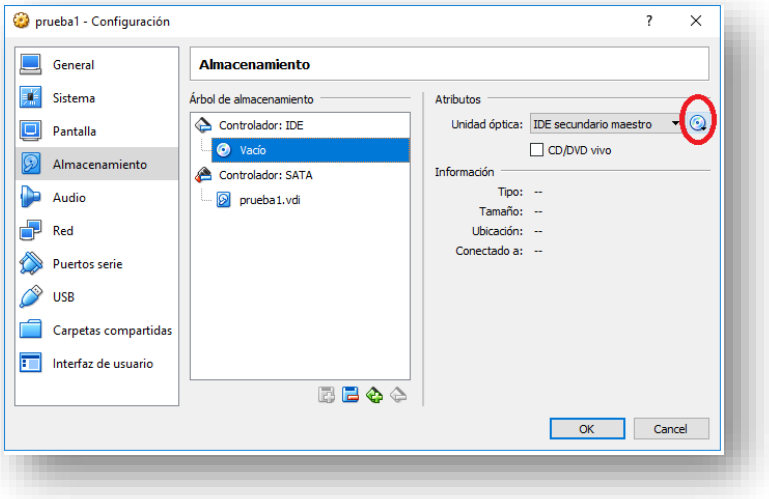

Imagen 28 – Selección de archivo .iso de arranque

En la pestaña Red, habrá que crear 2 redes, una tipo Nat para que la máquina virtual se conecte a internet, y otra Host-only para la conexión con el anfitrión:

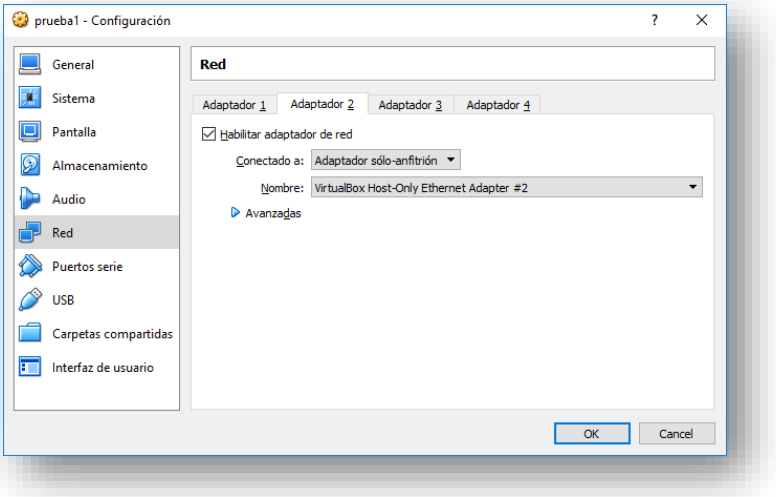

Imagen 29 – Configuración de la red de la máquina virtual

Una vez realizada la configuración, se inicia la máquina virtual, que arrancará con el sistema operativo de la imagen .iso que se le ha asignado.

Durante la instalación del sistema operativo, hay que seguir los siguientes pasos:

1. Seleccionar español como idioma.
2. Seleccionar español como idioma del teclado.
3. Como tipo de dispositivo, seleccionar Dispositivo de almacenamiento básico.
4. Darle un nombre al Host, y en la misma pantalla, pulsar en el botón configurar red aparecerá la siguiente pantalla:

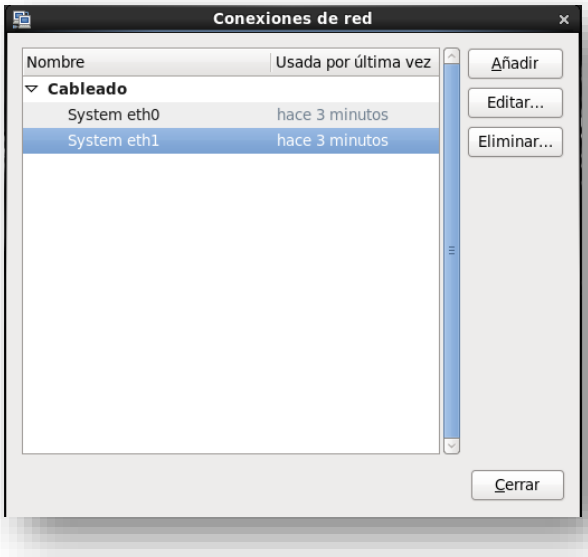

Imagen 30 – Configuración de la red de la máquina virtual II

En la imagen anterior se muestran las dos conexiones que tiene la máquina virtual, una para cada tipo de adaptador creado anteriormente. La conexión eth1 se editará, para que la gestión de direccionado ip4 sea manual, y se le asignará una IP fija, para poder acceder remotamente mediante ssh, una vez ya en marcha la máquina.

5. En la siguiente pantalla se selecciona la ciudad y se configura la hora.
6. Se establece una contraseña para el usuario root.
7. Y por último se elige el tipo de instalación, para este caso se elige la opción database server y se selecciona el paquete PostgreSQL, para que lo cargue en la instalación.

Una vez realizados los pasos anteriores, se procederá con la instalación del sistema operativo.

Como se ha apuntado anteriormente, el acceso a la máquina virtual se realizará remotamente desde el anfitrión. Para ello se va a utilizar el cliente ssh MobaXterm para Windows 10. Para conectarse, solo habrá que especificar la IP fija, otorgada en uno de los pasos anteriores, el usuario y la contraseña, con esto ya se podrá acceder a la máquina.

A la hora de conectarse a la máquina, pueden surgir una serie de problemas, uno de ellos es el bloqueo de la conexión por parte del firewall de Oracle Linux.

Para solucionarlo, hay que seguir las siguientes instrucciones:

1. Conectarse como root: su-.
2. Comprobar el estado del firewall mediante el comando *service iptables status*.
3. Desactivar el firewall con la sentencia *service iptables stop*.

Con esto se consigue desactivar el firewall a nivel de sesión, para hacerlo permanente, habrá que usar este otro comando *chkconfig iptables off*.

En caso de seguir teniendo problemas para acceder a la máquina, otra solución es desactivar el módulo de seguridad Selinux:

Para desactivarlo, hay que iniciar sesión como root y acceder al archivo '/etc/selinux/config' y editarlo, poniendo la variable *SELINUX a disabled*.

Anexo 2 – Manual básico de Psql para PostgreSQL

- **Como entrar en la base de datos:**
  psql -U myuser -h myhost "dbname=mydb

- **Listar las bases de datos desde psql:**
  \l

- **Ver las tablas pertenecientes a system:**
  select * from pg_tables where tableowner = 'postgres';

- **Listar las bases de datos desde la línea de comandos de Linux:**
  psql -U postgres -l

- **Descripción de una tabla:**
  \d nombre de la tabla

- **Cerrar psql:**
  \q

- **Conectarse a otra base de datos desde psql:**
  \connect databasename

- **Listar todas las tablas:**
  \dt

- **Listar todos los Schemas:**
  \dn

- **Listar todos los usuarios:**
  \du

- **Crear un nuevo usuario:**
  sudo -u postgres createuser nombre

- **Ver la versión de la Base de datos**:
  SELECT version();

- **Ver las conexiones a la base de datos:**
  SELECT * FROM pg_stat_activity;

Anexo 3 – Script de creación entorno test

```
Create table Propietarios
        (id_propietario Numeric NOT NULL,
        DNI Char(11) UNIQUE,
        nombre Text,
        apellidos Text,
        direccion Text,
        codigo_postal Numeric,
        provincia Text,
        telefono Numeric,
        primary key (id_propietario)
        ) ;
Create table Vehiculos
        (matricula Char(7) NOT NULL,
        bastidor Char(17) NOT NULL UNIQUE,
        marca Text,
        modelo Text,
        cilindrada Numeric,
        combustible Text,
        CV Numeric,
        tara Numeric,
        peso_maximo Numeric,
        id_propietario Numeric NOT NULL,
        primary key (matricula)
        ) ;
```

```
Create table Inspecciones
        (
        id_inspeccion Numeric NOT NULL,
        fecha_inspeccion Timestamp,
        KM Numeric,
        defectos_leves Text,
        defectos_graves Text,
        favorable Char(2) ,
        proxima_inspeccion Date,
        matricula Char(7) NOT NULL,
        id_estacion Numeric NOT NULL,
        primary key (id_inspeccion)
        ) Without Oids;
Create table Estaciones_ITV
        (
        id_estacion Numeric NOT NULL,
        nombre Text,
        direccion Text,
        codigo_postal Numeric,
        provincia Text,
        horario Text,
        email Text UNIQUE,
        primary key (id_estacion)
        ) Without Oids;
```

```
Create table Facturas
    (
    id_factura Numeric NOT NULL,
    fecha_factura Timestamp,
    concepto Text,
    base_imponible Numeric,
    Importe_total Numeric,
    id_inspeccion Numeric NOT NULL,
    primary key (id_factura)
    ) Without Oids;
Create table Infracciones
    (
    id_infraccion Numeric NOT NULL,
    fecha_infraccion Timestamp,
    descripcion Text,
    importe Numeric,
    pagada Char(2) ,
    fecha_pago Timestamp,
    matricula Char(7) NOT NULL,
    primary key (id_infraccion)
    ) Without Oids;
```

Alter table Vehiculos add foreign key (id_propietario) references Propietarios (id_propietario) on update restrict on delete restrict;

Alter table Inspecciones add foreign key (matricula) references Vehiculos (matricula) on update restrict on delete restrict;

Alter table Infracciones add foreign key (matricula) references Vehiculos (matricula) on update restrict on delete restrict;

Alter table Facturas add foreign key (id_inspeccion) references Inspecciones (id_inspeccion) on update restrict on delete restrict;

Alter table Inspecciones add foreign key (id_estacion) references Estaciones_ITV (id_estacion) on update restrict on delete restrict;

Bibliografía

[1]. https://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling

[2]. http://momjian.us/main/writings/pgsql/replication.pdf

[3]. https://wiki.postgresql.org/wiki/Shared_Storage

[4]. https://www.postgresql.org/docs/9.5/static/different-replication-solutions.html

[5]. https://wiki.postgresql.org/wiki/Direct_Storage_vs._SAN

[6]. https://www.postgresql.org/docs/current/static/hot-standby.html

[7]. https://www.postgresql.org/docs/9.1/static/auth-pg-hba-conf.html

[8]. https://www.postgresql.org/docs/9.1/static/runtime-config-connection.html

[9]. https://www.postgresql.org/docs/current/static/warm-standby.html#SYNCHRONOUS-REPLICATION

[10]. https://www.postgresql.org/docs/current/static/runtime-config-replication.html#GUC-SYNCHRONOUS-STANDBY-NAMES

[11]. https://wiki.postgresql.org/wiki/Slony

[12]. https://get.enterprisedb.com/docs/Tutorial_All_PP_Slony_Replication.pdf

[13]. https://www.pgadmin.org/docs/pgadmin3/1.22/slony-install.html

[14]. https://wiki.postgresql.org/wiki/Londiste_Tutorial

[15]. http://bdr-project.org/docs/1.0/weak-coupled-multimaster.html

[16]. http://bdr-project.org/docs/stable/

[17]. https://Bucardo.org/Bucardo/boolean/

[18]. http://www.dalibo.org/_media/pgpool.pdf

[19]. https://www.cpan.org/

[20]. Yair Amir, Claudiu Danilov, Michal Miskin-Amir, Jonathan Stanton, Ciprian Tutu PRACTICAL WIDE-AREA DATABASE REPLICATION

[21]. Sushant Goel, Rajkumar Buyya - DATA REPLICATION STRATEGIES IN WIDE AREA DISTRIBUTED SYSTEM

[22]. A. Silberschatz, H. Korth y S. Sudarshan. Database System Concepts. http://codex.cs.yale.edu/avi/db-book/db5/slide-dir/ch22.pdf

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR