Soluciones de Replicación en PostgreSQL 9.1

Objetivo

- Definir de forma simple y sintética algunos conceptos vinculados con la replicación.
- Introducir al alumno a la comprensión de las distintas técnicas de replicación que pueden implementarse en un producto como PostgreSQL 9.1

Requisitos

- Que el alumno comprenda los conceptos de transacción y tenga experiencia en la programación de sistemas de bases de datos.
- Que el alumno cuente con conocimientos de networking.
- Que el alumno cuente con conocimientos básicos de arquitectura monousuaria, file-server, client-server, n-capas.
- Que el alumno cuente con conocimientos básicos de sistemas operativos: file system, network file system, array disk, RAID.

1. Introducción

Cuando hablamos de *cluster* nos referimos a un conjunto de servidores que trabajan de manera coordinada; estos servidores están conectados en una red y hacia los usuarios del *cluster*, éste se muestra como si fuese un único servidor. Suele utilizarse esta técnica en servicios que requieren de *alta disponibilidad*, es decir, de que el servicio sea interrumpido lo menos posible, por lo general, se trata de servicios críticos y que se ejecutan las 24hs, los 365 días del año. Para alcanzar este requisito, ante una falla grave de un servidor, otro servidor del cluster rápidamente asume el rol del servidor que falló.

El trabajo coordinado de un conjunto de servidores puede aportar no sólo a la alta disponibilidad sino también al *balance de carga* del trabajo a realizar, por ejemplo, una consulta sobre una base de datos, podría ser resuelta por distintos servidores, según la carga de trabajo de cada uno, haciendo un uso inteligente del hardware disponible, se puede obtener un mayor rendimiento en sistemas con alta carga de trabajo. Varios sevidores podrían servir el mismo dato.

Un cluster podría estar formado por un conjunto de SGBD, una misma base de datos podría estar copiada total o parcialmente en distintos SGBD's, a esto lo llamamos una *base de datos replicada*. Responder a consultas sobre una base de datos replicada puede incrementar notablemente el rendimiento porque la consulta podría resolverse por el servidor más próximo al cliente y además, la capacidad de respuesta se incrementa por la cantidad de servidores disponibles.

Los queries o consultas sobre una base de datos pueden ser read/only (no modifican datos, son simples consultas) o read/write (modifican datos). Una transacción esta formada por un conjunto de queries read/write que deben ejecutarse de forma atómica: todo o nada; una transacción no puede ejecutarse por la mitad porque ello dejaría a la base de datos en un estado inconsistente, un estado no íntegro, no válido.

Es relativamente fácil coordinar varios servidores cuando se trata de queries *read/only*, el problema se presenta en un escenario de replicación cuando hay queries *read/write*, ya que, el dato modificado debe ser guardado en todas las copias de la base de datos y se debe servir el dato de forma consistente (el mismo valor), desde todos los servidores.

Algunas soluciones a este problema permiten que un solo servidor modifique los datos, a este servidor se lo llama *servidor maestro o master o primary server*. Hay otros servidores que siguen la pista de lo modificado por el servidor primario y replican el dato modificado, a estos servidores se los llama *servidor esclavo o standby o slave*.

Un servidor esclavo no puede ser conectado hasta que no sea promovido por el maestro, a estos servidores se los denomina *warm stadby server*. Un servidor esclavo que puede ser conectado y solo sirve queries read/only se lo denomina *hot standby server*.

Las soluciones propuestas a los problemas de replicación pueden clasificarse de distintas formas, por ejemplo, una *solución sincrónica* es aquella en donde el dato modificado no es considerado *comiteado* (aplicado a la base de datos, aceptado, *commited*) hasta que todos los servidores lo hayan comiteado. Todos los servers del cluster devuelven un mismo dato consistente. Una *solución asincrónica*, es aquella que permite una demora (delay) entre el commit del dato y su propagación a los demás servidores. Los servidores pueden devolver un dato no consistente, no necesariamente devolverán el mismo valor. Esta última opción se utiliza cuando la solución sincrónica es muy lenta debido al hardware disponible o bien al ancho de banda de la red.

Otras clasificaciones posibles podrían hacerse por granularidad: es decir, si se replica todo el server completo, o toda la base de datos completa, o toda una tabla completa.

2. Comparación de distintas soluciones

2.1 Shared Disk Failover

Evita la sobrecarga de la sincronización teniendo una sola copia de la base de datos utilizando un *arreglo de discos* compartido entre N servidores. Si el servidor maestro falla, un esclavo toma su lugar y comienza la recuperación de la base de datos a partir del fallo. Permite una rápida recuperación sin perdida de datos. Esta compartición es muy común en un ambiente de red, requiere de un *file system* de red con comportamiento POSIX, si falla el

arreglo de discos o hay corrupción en el mismo, todo fallará. Los esclavos no accederán al disco mientras el servidor maestro este operativo.

2.2 File System (Block-Device) Replication

Una modificación de la opción anterior, ahora hay un file system replicado, todos los cambios son espejados en otro file system de otra computadora. El espejado debe hacerse de forma tal que los esclavos tengan una copia consistente y debe hacerse en el mismo orden que lo hace el maestro. DRBD es una solución de replicación de file system muy popular en linux.

2.3 Warm and Hot Standby Using Point-In-Time Recovery (PITR)

Los servidores esclavos *warm standby* y *hot standby* pueden ser mantenidos actualizados leyendo desde un stream de tipo write-ahead log (WAL)[2]. Si falla el maestro, los esclavos tienen todos los datos del servidor maestro y puede promoverse un esclavo como nuevo maestro. Es una solución asincrónica y solo puede hacerse para toda la base de datos. Puede implementarse con *log shipping*[3] o streaming replication [4] (sección 25.2.5) o una combinación de ambos.

2.4 Trigger-Based Master-Standby Replication (Replicacion Maestro-Esclavo basada en triggers)

Una replicación maestro-esclavo envía todos los queries de actualización al maestro. Los esclavos son sólo de consulta. El maestro envía asincrónicamente todas las actualizaciones de datos a los esclavos. Los esclavos pueden atender queries de consulta. Los esclavos son ideales para queries de tipo *data warehouse*[5]. Slony-I es un ejemplo de implementación de este tipo de replicación, con una granularidad a nivel de tabla y soporta múltiples servidores esclavos (ya que los actualiza asincrónicamente, utilizando *batches* de actualización), es posible la perdida de datos durante un fallo.

2.5 Statement-Based Replicacion Middleware

Hay un proceso que intercepta todos los queries (Middleware) y los envía a uno o todos los servers. Cada servidor opera independientemente. Los queries read/write se envían a todos los servidores, para que todos reciban los cambios. Los queries read/only pueden ser enviados solo a un server, permitiendo distribuir la carga de trabajo entre los distintos servidores. Hay que tener cuidado con las funciones tales como random(), CURRENT_TIMESTAMP, etc. porque podrían tener valores diferentes en los distintos servidores; si esto no es aceptable, el middleware o la aplicación, deberá tomar estos valores de un servidor determinado y usar estos valores en los queries de actualización. Todas las transacciones deberán ser comiteadas en todos los servidores o bien deberán ser abortadas en

todos los servidores, para ello, se puede utilizar un protocolo de commit de 2 fases (2PC) usando PREPARE TRANSACTION y COMMIT PREPARED. Pgpool-II y Continuent Tugnsten son ejemplos de implementación de este tipo de replicación.

2.6 Asynchronous Multimaster Replication

Para servidores que no están conectados permanentemente, como el caso de una notebook o servidores remotos; mantener la consistencia de los datos entre todos los servidores es un desafío. En esta solución, cada servidor trabaja independientemente y periódicamente se comunica con otros servidores para identificar transacciones conflictivas[1]. Los conflictos pueden ser resueltos por los usuarios o por reglas de resolución de conflictos. Bucardo es un ejemplo de este tipo de replicación.

2.7 Synchronous Multimaster Replication

Cualquier servidor puede aceptar un query de actualización y antes de que la transacción sea comiteada, los datos modificados son transmitidos desde el servidor que acepto el query a todos los demás servidores. Un alto numero de actualizaciones pueden causar muchos lockeos degradando la performance (que muchas veces suele ser peor que si fuese un solo servidor). Los queries de read/only pueden ser ejecutados por cualquier servidor. Algunas implementaciones utilizan disco compartido para reducir la sobrecarga de la comunicación. Es la mejor opción para sistemas con muchas consultas y una tasa moderada de actualizaciones, no hay necesidad de distinguir entre maestro y esclavo, tampoco hay problema con funciones no determinísticas (ejemplo: random()) ya que se envían los datos modificados. Postgresql no ofrece este tipo de replicación, pero se puede usar el protocolo de commit de 2 fases de postgresql (PREPARE TRANSACTION y COMMIT PREPARED) para implementar esta solución en el código de la aplicación o en un servidor middleware.

1 Se refiere a 2 o más transacciones que, antes de ser aplicadas definitivamente (commit, comiteada), pretenden modificar un mismo item de dato (por ejemplo, el valor de un atributo en una tupla determinada en una relación determinada).

Característica	Soluciones Propuestas						
	2.1	2.2	2.3	2.4	2.5	2.6	2.7
Producto	NAS	DRBD	PITR	Slony	pgpool-II	Bucardo	
Método de comunicación	Shared disk	Disk blocks	WAL	tuplas	SQL	tuplas	Tuplas y lockeos en tuplas
No se requiere HW especial		X	X	X	X	X	X
Permite múltiples servers maestro					X	X	X
No hay sobrecarga de maestro	X		X		X		
No hay espera para múltiples servers	X		X	X		X	
No hay pérdida de datos por fallo en maestro	X	X			X		X
Esclavo acepta queries read/only			Sólo hot standby	X	X	X	X
Granularidad a nivel de tabla				X		X	X
No es necesario resolución de conflictos	X	X	X	X			X

Tabla 25.1 Matriz de Alta disponibilidad, balance de carga y replicación

Hay unas pocas soluciones que no caen dentro de las categorías anteriores:

2.8 Data Partitioning (Particionado de datos)

El particionado de datos parte las tablas en conjuntos de datos. Cada conjunto puede ser modificado por un solo server. Ejemplo: los datos se pueden partir entre las oficinas de Londres y Paris, contando con un servidor en cada oficina. Si se necesita hacer una consulta que combine datos de ambas oficinas, una aplicación puede consultar ambos servers o bien una replicación maestro-esclavo puede ser usado para mantener una copia read/only de los datos de la otra oficina en cada servidor.

2.9 Multiple-Server Parallel Query Execution (Ejecución de queries en paralelo con múltiples servers)

Muchas soluciones anteriores permiten que muchos servidores ejecuten muchos queries, pero ninguna permite que un mismo query utilice múltiples servers para que pueda

ejecutarse mas rápido. La idea es que varios servidores trabajen concurrentemente sobre un mismo query, partiendo los datos entre los servidores, cada uno ejecutando su parte y devolviendo los datos a un servidor central en donde todos los datos son combinados y devueltos al usuario. Pgpool-II tiene esa capacidad.

Referencias

[1] 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

[2] WAL se refiere a que todos los cambios son escritos primero en el log, guardando la imagen previa, permitiendo hacer tanto *redo* como *undo* en caso de fallos, con la información almacenada en el log. Permite implementar dos de las propiedades ACID que debe garantizar toda transacción: atomicidad y durabilidad.

Atomicidad: todo o nada, se hace toda la transacción o nada de ella, si falla en medio, todo vuelve para atrás.

Consistencia: completar una transacción, indica llevar a la base de datos de un estado consistente a otro estado consistente.

Isolation (aislacion): una transacción esta aislada del resto, corre como si fuese la única, como si estuviesen serializadas. El estado de la base de datos que se obtiene, es como si las transacciones se hubiesen corrido una detrás de la otra y no en forma concurrente.

Durabilidad: una vez que una transacción fue comiteada sus cambios perduran en el tiempo, no se pierden sus actualizaciones, queda almacenado en memoria secundaria, incluso aunque ocurran fallos: como un corte de energía, fallo de software, errores, etc.

Estado Consistente: base de datos que cumple con todas las reglas de integridad.

[3] log shipping, ver sección 25.2 del manual de PostgreSQL, en general, se conoce a esta técnica como hacer un backup del log de transacciones del servidor primario y transferirlo a los servidores esclavos, para que éstos hagan una recuperación (restore) en caliente (hot restore) de su base de datos en base al log recibido para “ponerse al día” de las transacciones aplicadas en el servidor maestro.

[4] streaming replication, ver sección 25.2.5 del manual de PostgreSQL, técnica similar a la descripta en [3], pero en este caso, a medida que se producen los registros de log en el servidor primario, éstos son enviados a los servidores esclavos para que éstos se “pongan al día” lo más rápido posible, sin esperar que un bloque o registro de log se llene por completo para ser enviado.

[5] Las bases de datos on-line, mantienen la información actual de un negocio, pero esta información, una vez que ya no es necesario su uso, porque dejo de ser “actual” y pasa a ser histórica, puede transferirse a un almacén general de datos históricos de la empresa o data warehouse, en donde allí se almacena en forma redundante y apto para realizar sobre dichos datos operaciones únicamente de consulta y de análisis y minería de datos, permitiendo, en base a información histórica, poder tomar decisiones de negocios.

Atte. Guillermo Cherencio.
UNLu – 11078 – BD II