Por qué la ingeniería de Uber cambió de Postgres a MySQL

Introducción

La idea consiste en explicar por qué la compañía Uber cambio de Postgres a MySQL de acuerdo a su crecimiento y las nuevas necesidades que se van generando con el pasar del tiempo, en el cual se describen las ventajas y desventajas de las diferentes aplicaciones de bases de datos. También hablan de las arquitecturas y versiones de las aplicaciones, dado el nivel transaccional como tuvieron que cambiar de aplicación de base de datos por unas reglas muy básicas como la integridad de la información, disponibilidad de los datos y confiabilidad de la información que entrega.

Desarrollo

Específicamente, en muchos de los casos en los que utilizamos Postgres anteriormente, ahora usamos Schemaless, una nueva capa de creación de bases de datos construida sobre MySQL.
La arquitectura de Postgres nos encontramos con muchas limitaciones de: arquitectura ineficiente para escribir, ineficaz replicación de datos, problemas con la corrupción de la tabla, pobre réplica de soporte MVCC y dificultad para actualizar a lanzamientos más nuevos.
Veremos todas estas limitaciones a través de un análisis de la representación de Postgres de los datos de tabla e índice en el disco, especialmente cuando se compara con la forma en que MySQL representa los mismos datos con su motor de almacenamiento InnoDB.
Uno de los aspectos principales del diseño de Postgres es la fila de datos inmutables. Estas filas inmutables se llaman "tuplas" en el lenguaje de Postgres. Las tablas mismas tienen índices, que están organizados como estructuras de datos (típicamente B-trees) que mapean campos de índice a una carga útil ctid.
Como mencionamos anteriormente, las tuplas de fila son inmutables. Este campo adicional permite a la base de datos determinar qué tupla de fila servir para una transacción que no puede ver la última versión de fila.

En la replicación cuando insertamos una nueva fila en una tabla, Postgres necesita replicarla si la replicación de transmisión está habilitada. El WAL representa un registro de los cambios que la base de datos planea realizar en los contenidos en disco de tablas e índices. Cuando el daemon Postgres se inicia por primera vez, el proceso compara los datos en este libro con los datos reales en el disco. Si el libro contiene datos que no se reflejan en el disco, la base de datos corrige cualquier tupla o índice de datos para reflejar los datos indicados por el WAL.
Postgres implementa la replicación de transmisión al enviar el WAL en la base de datos maestra a las réplicas. Si pausa una réplica principal y una réplica de Postgres cuando la réplica está totalmente capturada, el contenido real en el disco de la réplica coincide exactamente con el byte maestro de byte. Consecuencias del diseño de Postgres
El diseño de Postgres resultó en ineficiencias y dificultades para nuestros datos en Uber.

El mismo problema surge en Postgres. Sin embargo, estos índices todavía se deben actualizar con la creación de una nueva tupla de fila en la base de datos para el registro de fila.
En los casos en que la replicación de Postgres ocurre puramente dentro de un solo centro de datos, el ancho de banda de replicación puede no ser un problema. Sin embargo, cuando la replicación debe ocurrir entre los centros de datos, los problemas pueden escalar rápidamente. En este diseño, tuvimos una instancia maestra de Postgres (más réplicas) en nuestro centro de datos occidental y un conjunto de réplicas en el este.
La replicación en cascada limita los requisitos de ancho de banda del centro de datos a la cantidad de replicación requerida entre solo el maestro y una única réplica, incluso si hay muchas réplicas en el segundo centro de datos. Sin embargo, la verbosidad del protocolo de replicación de Postgres aún puede causar una cantidad abrumadora de datos para una base de datos que usa muchos índices. Este problema de ancho de banda también nos causó problemas con el archivo WAL. Nuevas réplicas de instantáneas de bases de datos. Corrupción de datos
Durante una promoción de base de datos maestra de rutina para aumentar la capacidad de la base de datos, nos encontramos con un error de Postgres 9.2. Las réplicas siguieron los cambios de la línea de tiempo de forma incorrecta, haciendo que algunos de ellos apliquen incorrectamente algunos registros WAL.
Por lo que pudimos ver, el problema solo se manifestó en unas pocas filas por base de datos, pero estábamos extremadamente preocupados porque, como la replicación ocurre en el nivel físico, podríamos terminar corrompiendo por completo nuestros índices de base de datos.
La réplica MVCC en Postgres no tiene una verdadera réplica de soporte MVCC. Este diseño plantea un problema para Uber.
Postgres necesita mantener una copia de las versiones de filas antiguas para MVCC. Si una réplica de transmisión tiene una transacción abierta, las actualizaciones de la base de datos se bloquean si afectan a las filas que la transacción mantiene abiertas. Por lo tanto, Postgres aplica un tiempo de espera en tales situaciones: si una transacción bloquea la aplicación WAL durante un período de tiempo determinado, Postgres elimina esa transacción.

La actualizaciones de Postgres son debido a que los registros de replicación funcionan en el nivel físico, no es posible replicar datos entre diferentes versiones de disponibilidad general de Postgres. Una base de datos maestra que ejecuta Postgres 9.3 no puede replicarse en una réplica que ejecuta Postgres 9.2, ni un maestro que ejecuta 9.2 se puede replicar en una réplica que ejecuta Postgres 9.3.

Al igual que Postgres, InnoDB admite funciones avanzadas como MVCC y datos mutables.
La diferencia arquitectónica más importante es que, mientras Postgres correlaciona directamente los registros de índice con las ubicaciones en disco, InnoDB mantiene una estructura secundaria. Por lo tanto, un índice secundario en MySQL asocia claves de índice con claves principales:
Para realizar una búsqueda de índice en el (primer, último) índice, realmente necesitamos hacer dos búsquedas. Sin embargo, debido a que los datos están normalizados, las actualizaciones de filas solo necesitan actualizar los registros de índice que la actualización de la fila realmente modificó. Además, InnoDB normalmente hace actualizaciones de fila en su lugar. Los datos de la fila anterior se copian en el segmento de retrotracción. No necesitamos actualizar estos índices, mientras que Postgres debería hacerlo.
En comparación, el proceso de Autovacuum de Postgres tiene que realizar exploraciones de tablas completas para identificar las filas eliminadas.

La replicación en MySQL admite múltiples modos de replicación diferentes:
La replicación basada en extractos replica instrucciones lógicas de SQL (p. Ej., Por otro lado, la replicación basada en filas, similar a la replicación WAL de Postgres, es más detallada, pero da como resultado actualizaciones más predecibles y eficientes en las réplicas.

En MySQL, solo el índice principal tiene un puntero a las compensaciones de filas en el disco. La secuencia de replicación de MySQL solo necesita contener información sobre las actualizaciones lógicas de las filas.
Por el contrario, la secuencia de replicación de Postgres contiene cambios físicos, como "En el desplazamiento del disco 8 382 491, escriba bytes XYZ". Con Postgres, cada cambio físico realizado en el disco debe incluirse en la secuencia WAL. Pequeños cambios lógicos (como actualizar una marca de tiempo) requieren muchos cambios en el disco: Postgres debe insertar la nueva tupla y actualizar todos los índices para apuntar a esa tupla. Por el contrario, la secuencia WAL de Postgres contiene cambios físicos en el disco, por lo que las réplicas de Postgres no pueden aplicar actualizaciones de replicación que entren en conflicto con las consultas de lectura, por lo que no pueden implementar MVCC.

Finalmente, la arquitectura de replicación de MySQL hace que sea trivial replicar entre diferentes versiones de MySQL. MySQL solo incrementa su versión si el formato de replicación cambia, lo cual es inusual entre varias versiones de MySQL. Otras ventajas de diseño de MySQL
Hasta ahora, nos hemos centrado en la arquitectura en disco para Postgres y MySQL. Por ejemplo, nuestras réplicas más grandes de Postgres tienen 768 GB de memoria disponible, pero solo unos 25 GB de esa memoria son en realidad fallas en la memoria RSS por los procesos de Postgres. Para buscar datos del disco, el proceso de Postgres emite llamadas al sistema lseek (2) y de lectura (2) para ubicar los datos.
Postgres, sin embargo, usa un diseño de proceso por conexión. Hemos tenido problemas significativos al escalar Postgres después de unos cientos de conexiones activas. Hoy en día, tenemos algunas instancias heredadas de Postgres, pero la mayoría de nuestras bases de datos están construidas sobre MySQL (normalmente usando nuestra capa Schemaless) o, en algunos casos especializados, con bases de datos NoSQL como Cassandra.

Conclusión

Logramos evidenciar como fue el cambio de Postgres a MySQL por los cambios y los nuevos requerimientos de la aplicación transaccional de Uber, para tener un mayor crecimiento y un rendimiento más óptimo. Se conoció como la capa de esquemas de MySQL y las conexiones concurrentes ayudaron a dar ese paso tan grande de cambiar de motor de base de datos.

La comparación entre MySQL y Postgres, parece aceptado que MySQL junto con Apache y PHP forman un buen equipo para servir páginas web con contenido dinámico, discusiones, noticias, etc., por ejemplo al estilo de SlashDot. En general, sistemas en los que la velocidad y el número de accesos concurrentes sea algo primordial, y la seguridad no sea muy importante (pueda bastar con hacer backups periódicos que se restaurarán tras una caida del servidor). En cambio, para sistemas

más serios en las que la consistencia de la BD sea fundamental (BD con información realmente importante, bancos, etc.) PostgreSQL es una mejor opción pese a su mayor lentitud.

De acuerdo a esto podemos tomar una decisión más acertada a la hora de elegir una aplicación de base de datos para utilizar eficientemente en nuestro proyecto que se llevara a cabo.