Resumen 5-6

> *Google's Spanner: Becoming a SQL System*

> *Fecha de entrega: 25/10/22 2:58 pm*

- Max Richard Lee Chung - 2019185076

1. Introduction

El Spanner de Google inició como un almacenamiento por llave-valor que ofrece multi transacciones, consistencia externa y failover transparente entre datacenters. En el transcurso de los últimos 7 años, se cambió a un sistema de bases de datos relacional para tener consultas robustas, en donde se cambió el tipo de arquitectura por el deseo de tener un comportamiento tradicional (ej. Bigtable) y las demandas de los procesadores de consultas. De esta manera, permite mantener una masiva escabilidad en la aplicación mientras que le ofrece a los clientes una plataforma poderosa. Actualmente se utiliza como un sistema de gestionamiento de base de datos OLTP para datos estructurados de Google y como beta como Cloud Spanner.

El procesador de consultas implementa un dialecto de Standard SQL, el cual es compartido por varios subsistemas de consultas dentro de Google. El Standard SQL está basado en el estándar ANSI SQL, usando características como ARRAY y tipo de fila (llamado STRUCT) para soportar datos anidados. Está hecho para servir una mezcla de trasacciones y cargas de trabajos analíticas y soporta consultas con baja latencia y tiempo de ejecución prolongado. El procesador se distribuye por sí mismo y usa técnicas de optimización estándar como el código de envío cercano a los datos, el procesamiento paralelo de partes de una sola solicitud en varias máquinas y la eliminación de particiones.

2. Background

Las transacciones de Spanner utilizan un registro de rehacer de escritura anticipada replicado y el algoritmo de consenso de Paxos se utiliza para que las réplicas estén de acuerdo con el contenido de cada entrada de registro. En particular, cada shard es asignado a un grupo de Paxos y cada grupo puede tener varios shards. La implementación usa un Multi-Paxos, en el que se elige un único líder de larga duración y puede confirmar varias entradas de registro en paralelo, para lograr un alto rendimiento y alta disponibilidad.

El control de concurrencia usa una combinación de bloqueo pesimista y marcas de tiempo. Para las transacciones de modificación-escritura de lectura y escritura ciegas, el bloqueo estricto de dos fases garantiza la serialización dentro de un grupo de Paxos dado, mientras que las confirmaciones de dos fases (en las que participan diferentes grupos de Paxos) garantizan la serialización en toda la base de datos. A cada transacción confirmada se le asigna una marca de tiempo, y en cualquier marca de tiempo T hay una único snapshot de base de datos correcta. Las lecturas se pueden realizar en transacciones de snapshot sin bloqueo, y todos los datos devueltos de todas las lecturas en la misma transacción del snapshot provienen de un snapshot coherente de la base de datos en una marca de tiempo especificada. Las lecturas obsoletas eligen una marca de tiempo en el pasado para aumentar la posibilidad de que la réplica cercana se actualice lo suficiente como para servir la lectura y las réplicas conservan varias versiones de datos. Las lecturas fuertes ven los efectos de

todas las transacciones comprometidas previamente por medio de lecturas de marca de tiempo. Es posible que las lecturas sólidas tengan que esperar a que la réplica cercana se recupere por completo, o pueden volver a intentarlo en una réplica más lejana que puede estar más actualizada.

Provee un framework RPC especial llamado framework coprocesador para esconder la complejidad de localizar datos. El coprocesador determina cual grupo de Paxos administra el dato abordado y busca la réplica más actualizada y cercana. Los shards se pueden mover entre grupos o ser divididos a otros más pequeños y el coprocesador desvia la solicitud al nuevo grupo.

Una réplica dada almacena datos en un sistema de archivos distribuido de solo agregar llamado Colossus. El almacenamiento se basa en árboles de combinación con estructura de registro, el cual soporta alto rendimiento de lectura. El formato original era un mapa ordenado llamado SSTables (sorted string table). Dado la naturaleza de la carga de trabajo de las consultas, los planes de ejecución son tratadas en diferentes lugares de Spanner.

Internamente, el compilador de consultas utiliza una unión correlacionada de operaciones para transformaciones algebraicas y operadores físicos. Estos operadores se usan en CrossApply (evalúa los mapas por cada tupla y los concatena) y OuterApply (emite una salida de una fila del dato de entrada). La ejecución de planes se encarga de enviar subplanes a servidores remotos y recopilar resultados para manejar ejecuciones particionadas.

Tanto el compilador como el tiempo de ejecución intentan realizar de forma agresiva la eliminación de particiones para minimizar la cantidad de trabajo realizado por las consultas al restringir los datos con los que operan sin cambiar los resultados.

3. Query distribution

Distributed query compilation

La unión distribuida se usa para enviar una subconsulta a cada shard de los datos persistentes o temporales subyacentes para concatenar los resultados. La distribución es fundamental para ubicar las tablas dado que en ejecución, puede estar en diferentes lugares. Se realiza un escaneo global de las tablas para reemplazar algún dato con escaneos internos de las tablas, ordenando los shards según la llave. La particionabilidad se debe de cumplir para empujar los datos hacia abajo operaciones básicas, sin embargo, conociendo las llaves o las identificaciones, se pueden empujar operaciones más complejas. También empuja operaciones que potencialmente toman como filas de entrada de múltiples shards.

Distributed execution

En tiempo de ejecución, la unión distribuida minimiza la latencia mediante el uso del framework de coprocesador Spanner para enrutar una solicitud de subconsulta dirigida a un shard a una de las réplicas más cercanas que pueden atender la solicitud. El recorte de fragmentos se utiliza para evitar consultar fragmentos irrelevantes, aprovecha las claves de rango de los fragmentos y depende de empujar hacia abajo las condiciones en las claves de los shards de las tablas para los escaneos subyacentes.

Antes de la ejecución de la subconsulta, realiza un análisis en tiempo de ejecución de la expresión del filtro de sharding keys para extraer un conjunto de rangos de sharding keys. El conjunto de rango extraído está garantizado para cubrir

completamente todas las filas de la tabla en las que la subconsulta puede generar resultados.

Cuando el rango de sharding keys extraído de la expresión de filtro cubre toda la tabla, la unión distribuida puede cambiar su estrategia de evaluación de subconsulta. En lugar de enviar la subconsulta a cada shard de la tabla y realizar una llamada entre máquinas por shard, la unión puede enviar una sola llamada a cada servidor que aloja un grupo de fragmentos que pertenecen a la tabla de destino. Dado que un grupo puede albergar varios fragmentos de la misma tabla, esta estrategia minimiza la cantidad de llamadas entre máquinas realizadas durante la evaluación de consultas.

Durante la ejecución, la unión puede detectar que los shards de destino están alojados localmente en el servidor y puede evitar realizar la llamada remota ejecutando su subconsulta localmente.

Distributed joins

Estilos aplicados de joins están correlacionados en las claves de la tabla funcionan muy bien en un entorno de procesamiento de consultas de una sola máquina donde la capa de almacenamiento admite búsquedas. Apply Join, Cross Join o Nested Loop Join en un ambiente distribuido implica una llamada entre máquinas realizada por cada fila procedente de la entrada izquierda del operador de combinación. Spanner implementa un operator de distribución aplicado extendido de la unión de distribución e implementando el estilo de aplicación de unión de forma conjunta.

La distribución aplicada permite minimizar el númeo de de llamadas entre máquinas para uniones basadas en claves y paralelizar la ejecución. Adenás, permite un recorte de shards con los siguientes pasos:

1. Evluar la expresión del filtro de clave del shard para cada fila del lote utilizando los valores de columna de cada fila.
2. Combinar los rangos de clave del shard para todas las filas del paso anterior en un solo conjunto.
3. Calcular el mínimo el conjunto mínimo de shards al que enviar el conjunto intersecando los rangos de claves de shard del paso 2 con los límites.
4. Construya un conjunto mínimo para cada shard mediante la intersección de los rangos de clave de shards del paso 2 con los límites de los fragmentos.

Query distribution APIs

Están diseñados para procesamiento de datos y sistemas de tipo de reducción de mapas que utilizan varias máquinas para unir datos de Spanner con datos de otros sistemas o para realizar transformaciones de datos fuera de Spanner.

- Single-consumer API: Usado cuando un proceso único de cliente consumme los resultados de una consulta.
- Parallel-consuer API: Usado para estar consumiendo resultados de consultas en paralelo de varios procesos en múltiples máquinas.

4. Query range extraction

Problem statement

La extracción de rango de consulta se refiere al proceso de analizar una consulta y determinar a qué partes de las tablas hace referencia la consulta.

- Extracción de rango distribuido: Saber a qué shards de tabla hace referencia la consulta es esencial para enrutar la consulta a los servidores que alojan esos fragmentos.
- Extracción de rango de búsqueda: Una vez que la consulta llega a un servidor Spanner, determinamos qué shards del shard relevante leer de la pila de almacenamiento subyacente. Según la estructura de las claves primarias y los tamaños de las tablas, la conversión de escaneos de fragmentos completos en búsquedas en rangos de claves más pequeños podría marcar una gran diferencia en el tiempo de ejecución de la consulta. Sin embargo, el costo de la extracción de rango puede superar el beneficio de convertir escaneos completos en búsquedas, por lo que los algoritmos eficientes son fundamentales.
- Extracción de rango bloqueado: Los rangos de claves extraídos determinan qué shards de la tabla se bloquearán (transacciones pesimistas) o se verificarán las posibles modificaciones pendientes (transacciones instantáneas). El método mejora la simultaneidad al permitir reemplazar un solo bloqueo de un rango grande con uno o más bloqueos detallados que cubren un subconjunto del rango grande. La compensación aquí es tener demasiadas cerraduras.

Compile-time rewriting

La implementación de la extracción de rangos en Spanner se basa en dos técnicas principales: en el momento de la compilación, normalizar y reescribir una expresión de exploración filtrada en un árbol de autocombinaciones correlacionadas que extraen los rangos para columnas clave sucesivas. En tiempo de ejecución, se utiliza una estructura de datos especial llamada árbol de filtro para calcular los rangos a través de la aritmética de intervalo ascendente y para una evaluación eficiente de las condiciones posteriores al filtrado. La reescritura en tiempo de compilación realiza una serie de pasos de normalización de expresiones, incluidos los siguientes:

- NOT se empuja a los predicados de hoja. Esta transformación es lineal en el tamaño de la expresión. No traducimos las condiciones del filtro a un DNF o CNF debido a la posibilidad de una explosión exponencial.
- Las hojas del árbol de predicados que hacen referencia a columnas clave se normalizan al aislar las referencias clave.
- Los intervalos de enteros pequeños se discretizan.
- Las condiciones complejas que contienen subconsultas o funciones de biblioteca costosas, aritmética, etc, se eliminan.

Filter tree

Se utiliza para extraer los rangos de claves a través de la intersección/unión de intervalos de abajo hacia arriba y para filtrar posteriormente las filas emitidas por las autouniones correlacionadas. Se comparte entre todos los análisis correlacionados producidos por la reescritura en tiempo de compilación. Memoriza los resultados de los predicados cuyos valores no han cambiado y reduce el cálculo del intervalo.

5. Query restarts

Usage scenarios and benefits

- Ocultar fallas transitorias: Esto significa que una transacción instantánea nunca devolverá un error en el que el cliente deba volver a intentarlo. La lista incluye errores genéricos, esperas de distribución y movimiento de datos internos.

- Modelo de programación más simple: sin bucles de reintento: Los bucles de reintento en el código del cliente de la base de datos son una fuente de errores difíciles de solucionar, ya que escribir un bucle de reintento con el retroceso adecuado no es trivial.
- Streaming de paginación a través de los resultados de la consulta: El código del cliente puede dejar de consumir los resultados de las consultas durante períodos prolongados (respetando la fecha límite de la solicitud) sin acaparar los recursos del servidor.
- Latencia de cola mejorada para solicitudes en línea: Importante para las cargas de trabajo de consultas analíticas y transaccionales combinadas que acceden a recursos considerables en paralelo en muchas máquinas.
- Reenviar progreso para consultas de ejecución prolongda:
- Actualizaciones continuas recurrentes.
- Manejo de errores internos más simple.

Contract and requirements

Ocupa un nuevo token de reinicio en el mecanismo RPC. Los tokens de reinicio acompañan a todos los resultados de la consulta, se envían en grupos, un token de reinicio por grupo. Un requisito para los tokens de reinicio es que sean pequeños en tamaño y baratos en rendimiento de CPU. Sin embargo, encuentra los siguientes retos:

- Resharding dinámico: Se debe de guardar el progreso del proceso en la llave de la tabla para garantizar el reinicio.
- No determinista: Hace que las filas de resultados se devuelvan en un orden no repetible.
- Se reinicia en todas las versiones del servidor: Los cambios entre versiones se deben abordar para preservar la capacidad de reinicio. (reinicie el formato del cable del token, plan de consulta y el comportamiento de los operadores)

Blockwise-columnar storage

Ressi data layout

Ressi organiza los datos en bloques en orden de fila principal pero presenta los datos dentro de un bloque en orden de columna principal. Ressi divide los valores en un archivo activo que contiene solo los valores más recientes y un archivo inactivo que puede contener muchas versiones anteriores. La estructura de datos fundamental de Ressi es el vector, que es una secuencia indexada de valores tipificados homogéneos. Dentro de un bloque, cada columna está representada por uno o múltiples vectores. Ressi puede operar directamente en vectores en forma comprimida.

Live migration from SSTables to Ressi

Requiere un cuidado extraordinario en la conversión para garantizar la integridad de los datos y una implementación mínima y reversible. Spanner es capaz de migraciones de datos en vivo a través de movimientos y transacciones de datos masivos. El mecanismo de movimiento de datos de Spanner copia los datos (y los convierte según sea necesario) del grupo actual, que continúa sirviendo datos en vivo desde un conjunto completo de réplicas. Luego de que todos los datos han llegado al nuevo grupo, puede asumir la responsabilidad de atender las solicitudes. El cambio de formato se impulsa gradualmente para minimizar el impacto en el tráfico en vivo.