Resumen 2 y 3

Max Richard Lee Chung - 2019185076

Introduction

Bigtable está diseñado para escalar con confianza a petabytes de datos y muchas otras máquinas. Ha adquirido varios logros tales como aplicabilidad universal, escalabilidad, alto rendimiento y alta disponibilidad. Se utiliza por su variedad de altas demandas de trabajo y configuraciones personalziadas parecido a una base de datos. Ofrece un sistema que permite un control dinámico sobre el diseño y el formato de los datos. Funciona de igual manera con índices de filas y columnas que permiten el formato de datos estructurados o semiestructurados, también los clientes pueden seleccionar los esquemas a utilizar.

Data Model

Bigtable es una escasa, distruibuida, persistene y multidimensional mapa ordenada, el cual está indexado por una row key, column key y una marca de tiempo en donde cada valor del mapa se encuentra almacenado en un arreglo de bytes no interpretados.

- Filas: Las llaves de la tabla son strings arbitrarios con una capacidad máximade 64KB de tamaño. Cada lectura o escritura bajo una sola llave es atómica para conocer bien el comportamiento del sistema en presenca de varias actualizaciones en la misma fila. Mantiene el orden de las llaves de forma lexicográfico. El rango de la tabla se realiza por medio de partición dinámica, llamada como "tablet"(unidad del load balancing). Como resultado, las lecturas de pequeños rangos son más eficientes y ocupa la comunicación con pocas máquinas. De esta forma, los usuarios pueden seleccionar las llaves para usar una buena localidad de los archivos para mayor eficiencia de acceso.
- Columnas: Las columnas son agrupadas en "column families" (unidad básica de control de acceso), en donde normalmente todos los datos son iguales. Estas columnas deben ser creadas antes para poder almacenar y posteriormente, usar los datos. Se intenta que sean pocas las distintas columnas que se utilicen en una tabla y que rara vez cambien durante ejecución. El nombre de una columna se basa en la familia a la cual pertenece y un calificatorio. Toda familia de columna debe de poder ser imprimible pero los calificadores pueden ser strings arbitrarios. El acceso de control y almacenamiento de disco y memoria se realiza en este nivel.
- Marcas de tiempos: Cada celda en la Bigtable puede contener múltiples versiones del mismo tipo de dato y estas versiones están indexadas por marcas de tiempos (enteros de 64 bits). Los tiempos pueden ser establecidos de forma automática (en microsegundos y que represetan el tiempo real) o de forma manual por medio de una aplicación del cliente. Aplicaciones que requieran evitar colisiones deben de generar marcas únicas de tiempo ellos mismos. Las versiones son almacenadas de forma descendiente para ver los más recientes primero. Para hacer que el manejo de las versiones de datos sea menos exigente, se puede realizar el uso de 2 configuraciones por familia de columnas para que el sistema active el recolector de basura automáticamente. Otra forma es que el cliente especifique mantener cierta cantidad de versiones para eliminar los más viejas.

API

Ofrece funciones para crear y eliminar tablas y familias de columnas, así como funciones para cambiar el cluster, tablas, meta datos de las columnas y los derechos de control de acceso. Las aplicaciones pueden escribir o eliminar valores, buscar valores de una fila o iterar un subcojunto de datos de la tabla. Bigtable soporta muchas otras características que permiten la manipulación de datos en muchas otras formas complejas, tales como:

- Transacciones de una fila que lleva a cabo una lectura, modificación y escritura atómica, ya que no soporta transacciones generales de varias llaves.
- Permite que las celdas puedan usarse como un contador de enteros.
- Permite la ejecución de guiones suministrados por el cliente en espacios habilitados del servidor.

Building Blocks

Utiliza la distribución del sistema de archivos de Google (GFS, Google File System) para almacenar archivos de datos y registros. Un cluster de Bigtable usualmente opera en un grupo de máquinas compartido que se encuentran en ejecución con una variedad de otras aplicaciones distribuidas y a veces los processos son compartidos entre las aplicaciones. Bigtable depende de un sistema de administración de clústeres para definir los trabajos, administración de recursos en máquinas compartidas, administración de errores y monitoreo de la máquina. Utilza el formato de archivo de Google SSTable, ya que ofrece un mapa persistente, inmutable y ordenada de las llaves a los valores. Las operaciones son proporcionados para buscar el valor asociado con una clave específica y para iterar sobre todos los pares clave/valor en un determinado rango clave. Internamente, cada SSTable contiene una secuencia de bloques (64KB pero es modificable) y un índice de bloques para localizar cada uno. De forma opcional, una SSTable puede ser completamente mappeado a memoria, el cual permite el escaneo sin tocar el disco. Bigtable utiliza un servicio distribuido persistente y con alta disponiblidad llamada "Chubby", en donde consiste de 5 réplicas (uno máster) y utiliza el algoritmo de "Paxos" para mantener las réplicas en caso de fallos. Los namespaces consisten en directorios o archivos pequeños. Cada directorio o archivo puede ser utilizado como lock. La librería de cliente ofrece almacenamiento en caché consistente de archivos Chubby. Cada cliente usa un servicio y cada sesión expira si no se puede actualizar en el tiempo establecido y pierde el lock.

Implementation

Se implemente con tres componentes principales: una librería contectada a todos los clientes, un servidor máster y muchos servidores tablas (pueden ser añadidas o eliminadas de forma dinámica). El máster es el responsable para asiganr tablas a un servidor de tablas, detectar la expiración de un servidor, balanceo de cargas de trabajo, recolector de basura en el GFS y maneja cambios en los esquemas. Los servidores de tablas manejan un conjunto de tablas. Los clientes interactúan directamente con los servidores y no con el máster.

- Table Location: Se usa la estructura de un árbol B+ de tres niveles. El primer nivel almacena archivos en Chubby que contienen la raíz de la tabla que contienen todas las localizaciones de las tablas en una tabla especial. Guardan la localización bajo una llave de fila codificada de la identificación y el final de la fila. La librería obtiene las localizaciones de forma secuencial y se dan tres casos diferentes: si no se sabe la localización se va a la

siguiente tabla, si se encuentra vacío se ocupa 4 viajes de ida y vuelta viajes y si la caché es obsoleto se toma 6 viajes de ida y vuelta. También se almacenan otros datos en las tablas especiales como los registros del historial.

- Table Assignment: Cada tabla es asignada a un servidor de tablas y el máster realiza un seguiminto de las tablas asignadas y no asignadas para hacer balance de carga de trabajo. Chubby inicializa y obtiene un lock únio en el directorio porque sin él, el servidor se detiene. Si lo anterior ocurre, el servidor trata de conseguir otro lock y si no lo encuntra, no puede volver a servir. El máster se da cuenta si un servidor está activo mediante consultas de estado y si está inactivo, mueve los pedidos a otro servidor no asignado. Cada vez iniciado el administrador del sistema clúster, se usa un único máster lock en Chubby, el máster escanea los directorios de los servidores, comunica todos los involucrados y escanea las tablas especiales para aprender los conjuntos de tablas. El máster monitorea todas las actividades como si se combinan dos tablas o se separan para juntar o dividir las cargas de trabajo.
- Tablet Serving: Almacena el estado persistente en GFS. Registra los registros en un historial por si es necesario recuperarlos en memoria (memtable) en orden SSTable. La recuperación de datos se realiza a partir de los datos de las tablas especiales de los punteros de rehacer para reconstruir el memtable. Cuando encuentra una función de escritura y escritura, verifica si está bien formado para luego autorizarlo.
- Compactions: Cuando se llena la memoria, se crea otro espacio de memoria. Los datos del bloque lleno se convierten a una SSTable escrito en GFS. Se realiza este proceso para reducir el espacio y uso de memoria. Luego de compactar muchos bloques a SSTables, se juntan algunos bloques para tener que realizar una lectura de una nueva SSTable (merging compaction para bloques pequeños y major compaction para una sola SSTable).

Refinements

Los siguientes refinamientos es para conseguir alto rendimiento, disponibilidad y confiabilidad requerida.

- Locality groups: Grupos de familias de columnnas. Cada grupo tiene su propio SSTable en cada tablet. Las familias de columnas segregadas que no son típicamente accedidas al mismo tiempo son más eficientes. Además, se pueden personalizar los parámetros básicos de cada grupo. Una vez creadas, se pueden leer sin acceder al disco.
- Compression: Los usuarios pueden optar si comprimir las SSTables o no, en donde se puede seleccionar cuál formato usar. Hay dos formas de comprimnir de forma rápido, el esquema de Bentley and MacIlroy busca comprimir strings comunes largos y un algoritmo de compresión rápida que busca repeteciones en pequeñas tablas de 16KB.
- Caching for read performance: Con el fin de mejorar el rendimiento, los servidores de tablas están divididas en dos niveles de almacenamiento caché. El "Scan Cache" esta en un alto nivel y se usa para leer la llave de valor de la SSTable. El "Block Cache" esta en un bajo nivel que lee los bloques de SSTable que fueron leídos desde el GFS.
- Bloom filters: Si no se encuentra una SSTable en memoria, se puede optar a que el usuario seleccione un Bloom Filter. Un Bloom Filter permite preguntar si una SSTAble tiene la información especificada. Este proceso puede reducir el número de lecturas a disco para las operaciones de lectura y que no toquen el disco.

- Commit-log implementation: Usar sólo un historial de registro puede mejorar considerablemente el rendimiento pero complica mucho la recuperación de datos. La recuperación requiere muchos proceso para llevar a cabo la acción, por lo que una solución es que sólo se recuperen los registros necesarios para el funncionamiento adecuado del servidor pero se tendría que leer dependiendo de la n cantida de máquinas. Como evitamos que los registros sean duplicados, se ordenan los registros por orden numérico para leerlos de forma secuencial y lineal. Como medida de protección, se tienen 2 hilos pero sólo uno se encuentra activo.
- Speeding up tablet recovery: Se comprime las primeras tablas para reducir el tiempo de recuperación por la reducción de los estados descomprimidos. Luego vuelve a comprimir para eliminar restos descomprimidos y luego se sube a otro servidor.
- Exploiting immutability: La únca estructura mutable es el memtable. Si el SSTable se elimina, se considera como basura obsoletas. El beneficio de ser inmutables es que podemos dividir las tablas rápido y simplificado.

Performance Evaluation

Usando la misma cantidad de servidores (1GB en GFS de 1786 máquinas) con la misma cantidad de Bigtables con la misma cantida de clientes máquina dieron como resultado el mismo benchmark similar con respecto a la secuencia de lectura y escritura con operaciones opuestas a cada una. El benchmark de escaneo también fue similar a la secuencia de lectura pero recibió apoyo del API de Bigtable. Las lecturas aleatorias fue similar al benchmark de lectura pero los grupos contenían información en memoria que tomaban las lecturas del servidor.

- Single tablet-server performance: Las lecturas son mucho más lentas en servidor, sin embargo las lecturas aleatorias de memoria son más rápidas. Si los Bigtables tienen esta configuración, se tratan de hacer bloques pequeños para tener datos más reducidos. Además, no hay mucha diferencia con las escrituras aleatorias y secuenciales, ya que se agregan al mismo historial de registro y se empacan para ser escritas en GFS. Los escaneos secuenciales tienen mejor rendimiento que las lecturas.
- Scaling: Al incrementar la cantidad de sistemas, el rendimiento será mayor y efectivo, sin embargo no mejora de forma lineal ya que se llega a cierta capacidad máxima debido al CPU y/o red.

Real Aplications

1. Google Analytics: Análisis de patrones en páginas webs. Ofrece estadísticas de uso de visitantes únicos y vistas por día.
2. Googel Earth: Permite al usuario navegar por la superficie de la tierra desde una interfaz de mapa.
3. Personilized Search: Almacena registros de búsqueda y "clicks" en Google para hacer un historial cronológico y poder visitar de nuevo esas páginas.

Lessons

Los largos sistemas distribuidos son vulnerables a mmuchos tipos de fallos y se cambiaron varios protocolos para contrarrestarlos. Mejor atrasar nuevas características hasta conocerlas muy bien cómo usarlas según las necesidades requeridas. También, es importante tener un buen sistema de monitoreo para poder detectar y arreglar los problemas de forma inmediata. Por último, tener un diseño simple ayuda mucho a la hora de realizar manteniiento y depuración.