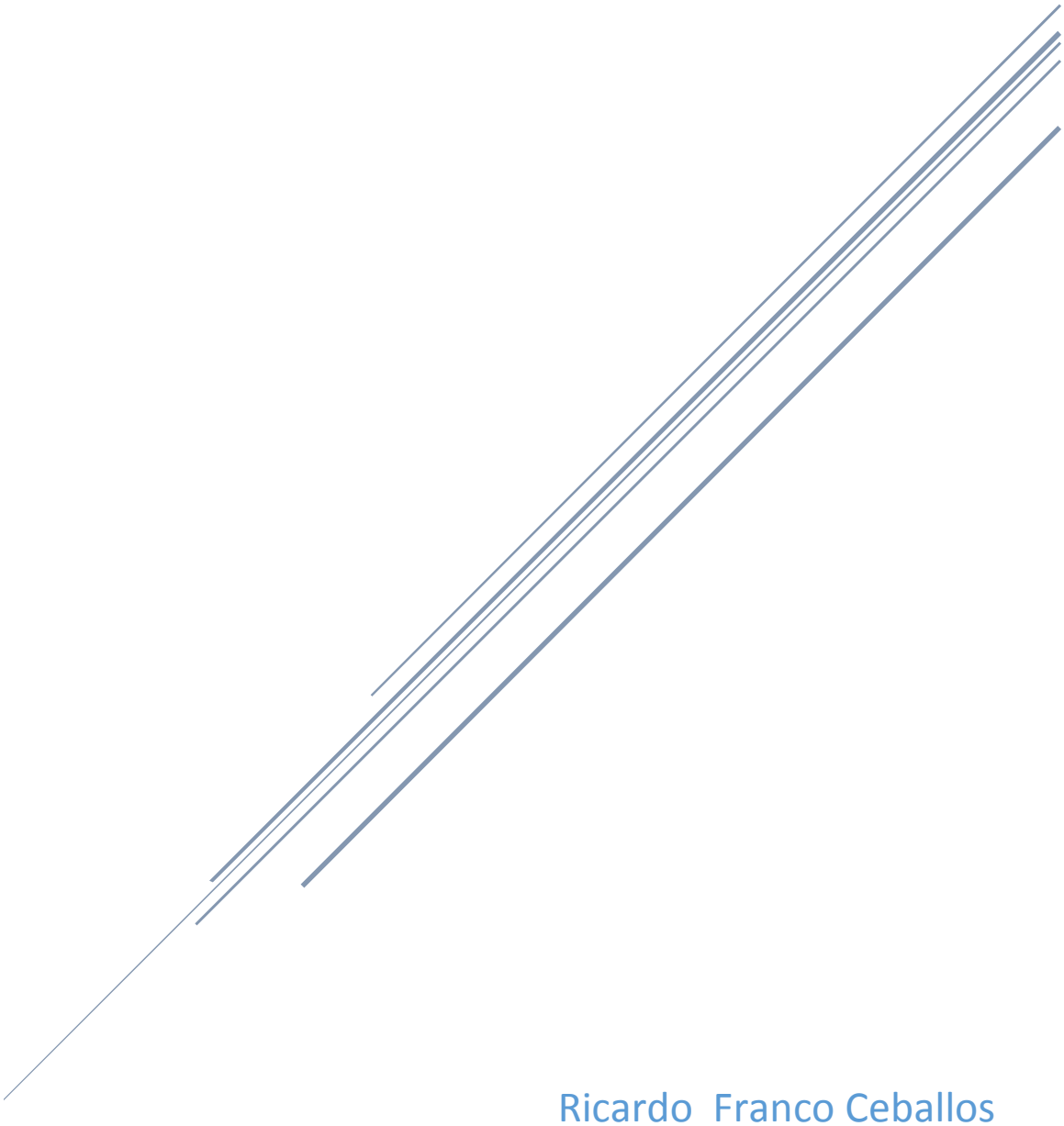
UBER Y SU MALA ELECCIÓN

ENSAYO



Ricardo Franco Ceballos
Juan Camilo Martínez Arroyave

Desde la creación de Uber, el esquema del software ha cambiado constantemente, buscando las mejorías en su aplicación y aumentar el rendimiento y la escalabilidad del sistema, optaron por dejar de usar un diseño monolítico basado en Postgres, a usar una nueva capa de construcción basada en MySQL.

El cambio surgió debido a las limitaciones que Postgres posee, básicamente estas limitaciones afectaban el funcionamiento del aplicativo y era un trabajo tedioso poder brindar datos confiables y realizar actualizaciones, además que Postgres al parecer se quedó estancado en el tiempo, sin lanzamientos de nuevas versiones con cambios significativos en su arquitectura ni en su sistema.

Antes de todo, hablemos un poco de Postgres y su arquitectura, Postgres maneja una especie de clave primaria por tuplas de datos llamada ctid, este ctid representa la ubicación del dato en el disco físico, todo esto para tener un control de corrupción en los datos almacenados, esto es trasparente para el usuario, pero te puede ayudar para comprender el diseño y el motor de base de datos.

En el ejemplo utilizado en el artículo, se utilizó una tabla con los campos Id, primero y segundo que equivalen al nombre del usuario y fecha de nacimiento, a su vez se crearon índices para esta tabla básica. Como se dijo anteriormente cada tupla tiene un ctid en un espacio de disco físico, por lo tanto, en el momento de actualizar un registro en la tabla, Postgres lo tomaría como si fuera un nuevo registro y esto va a complicar las cosas.

Postgres utiliza una replicación de datos, que se implementa en cuando se inserta una nueva fila en una tabla, el método consiste en habilitar la replicación de transmisión por si sucede un fallo con esto se puede recuperar, además Utiliza un WAL que lo que hace es que almacena en un espacio de disco, datos que no fueron exitosamente almacenados en la tabla, esto con el fin de poder reponerse y realizar una comparación tanto de los datos WAL como los originales de la tabla, esto con el fin de dar continuidad a las transacciones comprometidas durante el fallo, este sistema es verdaderamente útil para recuperar datos durante un proceso.

Postgres es un buen motor pero dependiendo del aplicativo se aconsejaría analizar bien cual escoger, para el caso Uber, Postgres se quedó corto, como se había dicho anteriormente, al momento de realizar una actualización de un campo este proceso se ve engorroso por parte de Postgres, debido que este lo interpreta como si fuera un nuevo registro, esto quiere decir que a pesar de que se actualice un solo campo, los demás campos den el registro también sufrirán una actualización, como se lee en el artículo, si se posee muchos campos en una tabla, y cada campo posee un índice, aunque solo se actualice un solo campo, este buscara actualizar también los índices de todos los registros, y esto es producto del ctid, debido a que el espacio

del registro actualizado cambio y los índices buscaran apuntar al más reciente, ahora no solo se modificara este tablespace sino también el tablespace del WAL y todos los tablespace involucrados, solo por actualizar un solo campo, y si este caso se refleja en una tabla con N campos y con N registro con una concurrencia alta y con actualizaciones constantes, ralentizara los procesos y bajara el rendimiento de la base de datos, haciéndolo de un proceso tedioso y engorroso.

No obstante también fue un desafío, mantener el rendimiento de banda ancha entre los diferentes servidores y replicas y el rendimiento y consumo de Postgres por parte del sistema WAL, como se tiene una mayor concurrencia y el consumo aumenta, el consumo de ancho de banda se verá demasiadamente afectado y solo en un país, ahora para intentar conectar varios países y poder tener el sistema actualizado bajo el WAL, el consumo puede desbordar y causar deficiencias y bastante lento, además fue un problema la capacidad del disco debido al crecimiento de los datos WAL.

Un problema que afronto Uber bajo Postgres fue la corrupción de datos, dado que se tenía varias réplicas de la base de datos que los servicio consumirían, al momento de actualizar un valor en una tabla determinada, esto en si se vería como un nuevo registro, hasta ahora todo normal, pero al actualizar Postgres a una versión más nueva, ocurrió el error de que no se podía validar que campo era el correcto, esto afecto el funcionamiento y arrojo datos erróneos, afortunadamente esto no afecto al Master y se pudo reconstruir las réplicas con base a este, y reparar el daño con ayuda de código, se mitigo el error, pero bajo un costo grande, la integridad de los datos es lo más importante en el sistema y que esto se vea comprometido por el motor de la base de datos, deja mucho que desear.

Otro problema al usar Postgres es la copia de seguridad, en si este no tiene una copia de seguridad sino que se ayuda con WAL y estos registros son almacenados a nivel físico, pero esta copia no siempre es efectiva ni rápida, debido que no se va realizar la copia hasta que una transacción finalice por completo, es decir que si la transacción se deja abierta este no podrá finalizar la copia de seguridad, el ejemplo que nos dan es que si existe un evento en el cual se realiza un envió de la factura al correo electrónico del cliente y que este activa una transacción y finaliza solo cuando el correo sea enviado, posiblemente este tarde, ahora bien si se piensa en gran tamaño, y que son miles de usuarios a los que se les está enviando la factura por correo, la transacción no se está cerrando y por ende no se puede realizar la copia de seguridad, un problema bastante grande para Uber.

También se evidencio el problema de versiones de Postgres al momento de intentar actualizarse, resulta que las réplicas no se pueden replicar en diferentes versiones de Postgres, así que si se manejan replicas con versiones de Postgres diferentes, es un trabajo engorroso y demasiado lento para poderlos actualizar, tanto así que se debe a pagar por completo el maestro y esto implica darle baja a la aplicación por un tiempo indeterminado, además que si se posee un gran tamaño y crecimiento

de datos, este tiempo aumentara. Podemos dar a la conclusión que es un proceso demasiado complicado y riesgoso, que compromete la integridad de los datos, además que es lento y que deja fuera de servicio el sistema, se puede decir que da a comprender que es mejor no actualizar las versiones del motor y estancarse en el tiempo.

Debido a todos estos problemas, fallos y desventajas producidos por Postgres, Uber opto por adaptar su sistema a MySQL.

La primera diferencia de MySQL y Postgres es que a diferencia de Postgres que tiene un puntero ctid por cada tupla que apunta a un espacio de memoria, en MySQL no existe esto, cada indexado apunta a la clave primaria, esto quiere decir que si se ejecuta un índice compuesto, primero tendrá que analizar el primero campo y luego el segundo, pero al actualizar un campo solo se alterara el campo referente y el índice que apuntador, lo contrario pasaba con Postgres que se modificaba todo el registro y todos los índices, esto es una ventaja que agiliza los procesos y rendimiento de la base de datos.

Otra diferencia es la replicación, como vimos anteriormente, este proceso en Postgres se realizaba por medio de WAL que almacenaba en el disco y que si se modificaba un solo campo se modificaba los punteros de todos los índices, en otra parte MySQL ofrece tres tipos de replicación, los cuales son más compactos y un proceso menos tedioso y más rápido, además la arquitectura de MySQL es poco propensa a que ocurran fallos en estos proceso de réplica, esto reduce la corrupción de los datos que tanto afecta a Postgres.

Además, anteriormente se habló que para actualizar versiones de Postgres y emparejar las copias era un trabajo muy duro y largo, con MySQL estas versiones siguen un mismo esquema lo que hace que los formatos de réplicas sean compatibles entre otros, además para actualizarlo se necesita de un tiempo de desconexión casi nulo, lo contrario pasaba con Postgres que podían pasar horas y que dependiendo de la densidad de la base de datos, este tiempo podría aumentar.

Otra ventaja de usar MySQL es el manejo de la conexión, el cual se realiza por hilos, estos hilos al finalizar un proceso se cortan y dan espacio a nuevas conexiones, realizando un consumo mucho menor y rápido, por lo contrario de Postgres, el cual realiza una conexión por proceso el cual es mucho más costoso que por hilo.

Es notable el cambio que surgió al momento de cambiar de Postgres a MySQL, pero aun en Uber se sigue utilizando esta. Se podría decir que antes de diseñar una base de datos o de implementar un sistema el cual vaya a necesitar de un motor de base de datos, se tiene que pautar los alcances, características y nivel de concurrencia entre otros, además de validar las diferentes opciones que existen de motores de base de datos, y ver qué ventajas y desventajas se presentan implementando ese sistema al nuestro, también es bueno pensar en futuro, y realizar simulaciones de eventos que puedan ocurrir en un tiempo, como el caso de las actualizaciones de

versiones, manejo de índices, conectividad y la fabricación de copias de seguridad, estos son factores claves que pueden inferir en un buen motor para nuestro sistema y que sin duda, es la base para que el sistema sea productivo.