UBER journey from PostgreSQL to MYSQL

Abstract: Nothing is perfect there is always a scope of improvement. Technology is changing every quarter. Improvement is the only way which can put a business in a spot to compete with their competitors and grow in the market. This article is about Uber Technologies which is an American multinational online transportation network company [1]. How it moved from PostgreSQL to MYSQL and how the database structure changed and helped them.
This article is based o UBER experience with PostgreSQL 9.2 release.

Introduction: Projects related to database and transaction processing always do fail. But the point to consider is how often do they fail. According to a report published in 1995, "Only 9% of projects were successful in large companies whereas 16.2% and 28% respectively at medium and small companies" [2]. Failure results in the loss of money, resources and time. Hopefully not that often the failures happen and there is always a better way to improve the technology to utilize the tools to benefit at the most.

Architecture of PostgreSQL:

PostgreSQL is an object-relational database system(ORDBMS) as well an open source equipped with the feature of traditional proprietary database systems and also have next generation DBMS systems. Its primary function is to store data securely and retrieval of data on request. It can handle small single machine applications to large Internet-facing applications as well supports all the modern Unix-compatible platform.

As PostgreSQL is a relational database so it is expected to perform some key tasks:

- Insert/Update delete
- Schema changes
- Implementation of MVCC (Multiversion concurrency control) to have a transactional view of data for different connections.

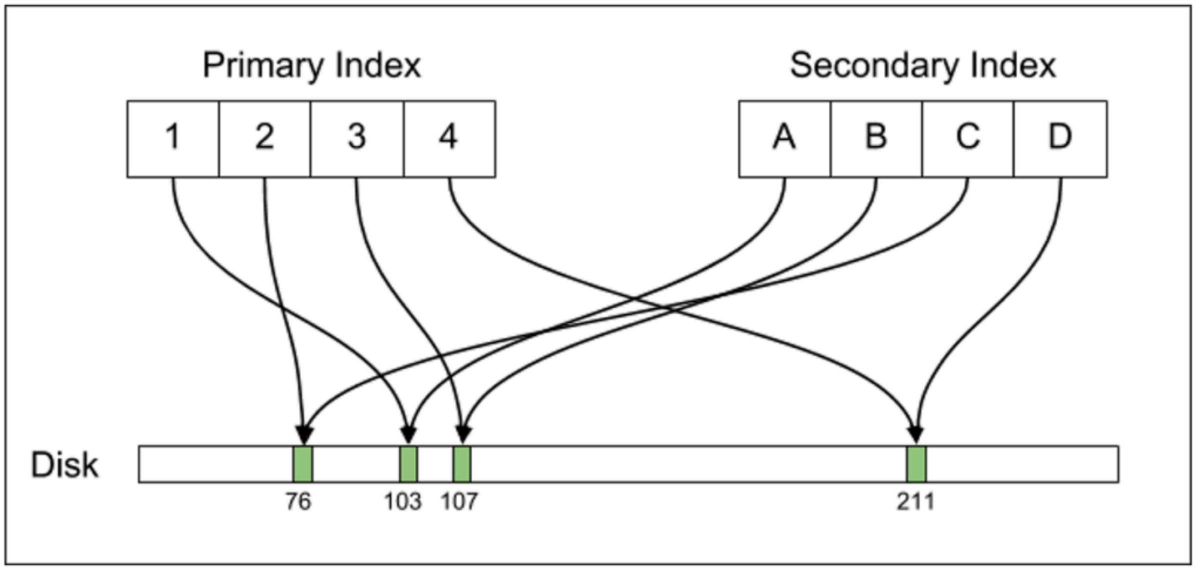
All of the above properties depends on the data which is on the disk.
Let us look at the on-disk location which is also called ctid in PostgreSQL. PostgreSQL's core design aspects have immutable row data where multiple version of the row exist for MVCC or when old versions of a row have not yet been reclaimed by the auto vacuum process.

For example, if we have to update the age of a user which is already existing

First name	***Last name***	***ctid***
Hilary	***Swank***	***G***
Tom	***Cruise***	***D***
Tom	***Cruise***	***F***
Steve	***Martin***	***M***

Age	***ctid***
30	***G***
35	***D***
36	***F***
42	***M***

Red represents the updated version and yellow old version
Added field help the database to determine the tuple need to serve for the transaction



Primary index and secondary indexes all point to the on-disk tuple offsets. Indexes must be updated with tuple changes.

Replication:

Whenever a new row is inserted into a table, PostgreSQL should replicate it due to streaming enable feature. The database maintains a WAL (write-ahead-log) which is used for 2 phase commit. WAL is maintained even if streaming feature is disabled to allow the atomicity and durability feature of ACID. The WAL represents a ledger of the changes the database plans to make to the on-disk contents of tables and indexes. When the PostgreSQL daemon first starts up, the process compares the data in this ledger with the actual data on disk. If any mismatches are present, the WAL updates that data.

As the WAL is designed for recovery purpose, it contains only the low level information about the on-disk updates. "Rsync" tolls are fixed to fix corrupted replica if WAL data is out of date with the master.

PostgreSQL failures affecting Uber:

Due to PostgreSQL design consequences, inefficiencies and difficulties were brought up for Uber users.

1) Write Amplification –
   To make a small change, we need to have 4 physical updates. Referring to the above example for age update, we need to have these 4 changes:
   - Write new row to table
   - Primary key index update for new row
   - Update the name
   - Update the age.

   Problem: Primary key update and name update: even if we have to update only the age, we have to update both the components for a small change. Hence for a single row update all indexes should be updated to reflect the ctid for the new row.

2) Replication –
   Write amplification problem translated to replication problem as all the four updates need to be updated into the replication layer resulting in occupying large amount of bandwidth.

For instance, Uber originally used physical servers in a colocation space on the West Coast. For disaster recovery purposes, we added servers in a second East Coast colocation space. In this design we had a master PostgreSQL instance (plus replicas) in our western data center and a set of replicas in the eastern one.

Cascading replication- It limits the center bandwidth requirements to the amount of the replication required. Cross-country links are expensive as well difficult to get link with same bandwidth as local connection. For Uber, sending all WAL updates from West to East Coast was problematic due to bandwidth issues. The issue was for both WAL archival and storage for disaster recovery. As well, Uber's bandwidth speed was not that much during peak traffic hours to keep up with the rate at which WALs were written.

3) Data corruption –
The replicas followed timeline switches incorrectly, due to this bug records were marked inactive which wee not supposed to. For example, as per the above example if we fire a select query, we will get two records and if ctid will be mention in the **Where** list then it would evidently show two records. It was difficult for Uber to figure out the duplicated rows and hence they ended up using defensive programming statements to figure out the situation.
Whereas the bug affected all the servers so the tables presented in all other servers were also affected row of one table was affected whereas Y row of another table was affected and X was good but it became difficult for them to figure out the real corrupted records. As well if they remove wrong data then the B-trees can become invalid.
Uber was able to figure out the actual bug and found that the newly promoted master did not have any corrupted rows. The issue was fixed by resyncing all from a new snapshot of the master.

A new version of PostgreSQL could be released at any time that has a bug of this nature, and because of the way replication works, this issue has the potential to spread into all of the databases in a replication hierarchy.

4) Replica MVCC –
PostgreSQL does not have replica model. They create a copy of on-disk data similar to master. PostgreSQL has timeout schedule which kills the transaction if one exceeds the time threshold. It pauses the WAL application thread which can cause the replica lag behind its master.

Due to lack of true replica MVCC support, hence it results in having a copy of on-disk data identical to the master at any point of time causing issue for Uber.

There were a lot of steps followed for PostgreSQL upgrade from 9.2 to 9.3 but it was so time taking that Uber was not able to again relocate time and resources on the up gradation. By the time the real PostgreSQL version 9.3 came out, Uber's database was increased a lot.

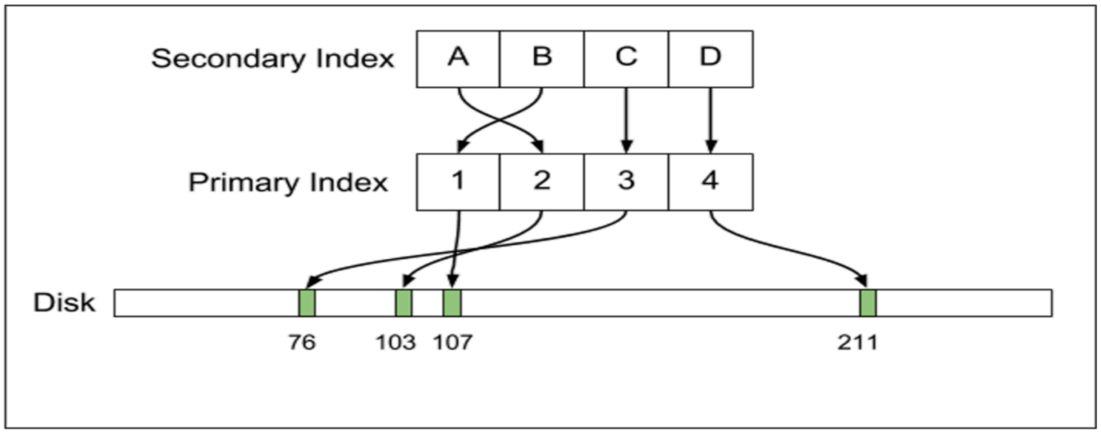
Limitations encountered:

- Unproductive architecture for writes
- Unproductive data replication
- Issues with table corruption
- Poor replica MVCC support
- Difficulty upgrading to newer releases

MySQL to the rescue:

To maintain a schemaless storage project, MYSQL is an important tool for Uber. Not just because it overcomes the above issues, MYSQL have its own benefits as well which made Uber to chose it over PostgreSQL.

1) *InnoDB On-Disk Replication*:
   It supports features like MVCC and mutable data.
   In MySQL, InnoDB maintains a secondary structure instead like PostgreSQL which used to point to ctid (disk row location), InnoDB secondary location index hold a pointer to the primary key value. Hence secondary index associates index keys with primary keys.



2) *Replication*:
   Different replication modes are supported by MYSQL:
   - Statement-based replication- Replicated logical SQL statements.
   - Row based replication- Alters row
   - Mixed replication – Includes both

3) *Other MYSQL disadvantages* –
   These are the additional advantages which gives upper edge to MYSQL compared to PostgreSQL.
   - The Buffer Pool –
     PostgreSQL used to have page cache which had its own disadvantages.
     Whereas MYSQL InnoDB buffer pool has some huge advantages:
     a) Helps to implement custom LRU design which helps in detecting pathological access patterns. This helps in preventing too much damage.
     b) Fewer context switches which doesn't require any user/kernel context switches

   - Connection Handling-
     Thread-per connection is used by MYSQL to implement concurrent connections. Threads usually have memory overhead for stack space.

MYSQL have some features which are new as well far better when compared to existing PostgreSQL features.

Conclusion:

PostgreSQL was used by Uber during their early days but due to some bug and upscaling of database Uber end up using MYSQL. Most for Uber databases are built on MYSQL basically Schemaless layer. Uber still have legacy PostgreSQL instances but due to the advantages and the way MYSQL benefited them, Uber is primarily dependent on MYSQL.

References:

1) https://en.wikipedia.org/wiki/Uber_(company)
2) http://dssresources.com/faq/index.php?action=artikel&id=282
3) https://eng.uber.com/mysql-migration/