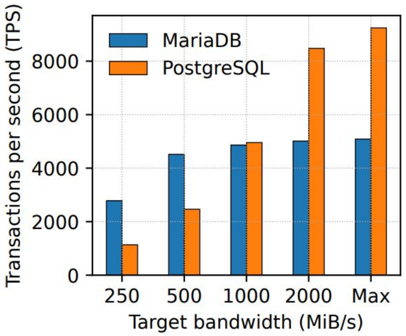
Analyzing Performance Characteristics of PostgreSQL and MariaDB on NVMeVirt

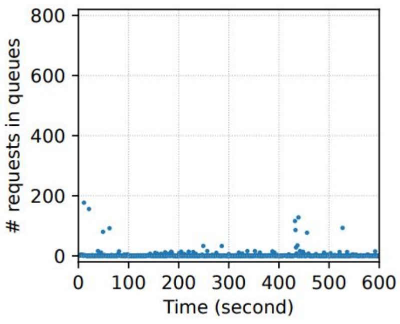
2022-23583 최유진, 2022-29677 한주희

Introduction

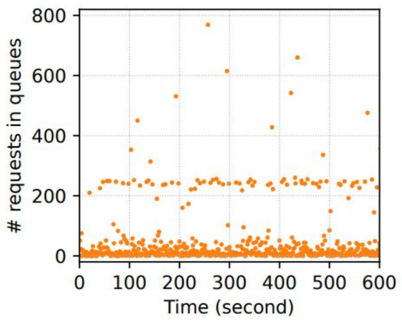
- I/O bandwidth limit influences both database engines, but PostgreSQL is much more sensitive than MariaDB.
  ➝ NVMeVirt paper concluded that *PostgreSQL is more promising on modern storage devices, whereas MariaDB is more efficient when the storage is slow.*
- *Problem* is that the paper does not provide a clear explanation of why there exists such differences.
- *Our goal* is to analyze the result with a focus on what features of database engine internals make such differences.



(c) TPS vs. target bandwidth



(e) Queue depth (MariaDB)



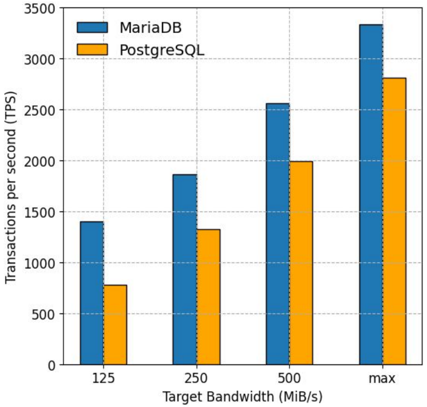
(f) Queue depth (PostgreSQL)

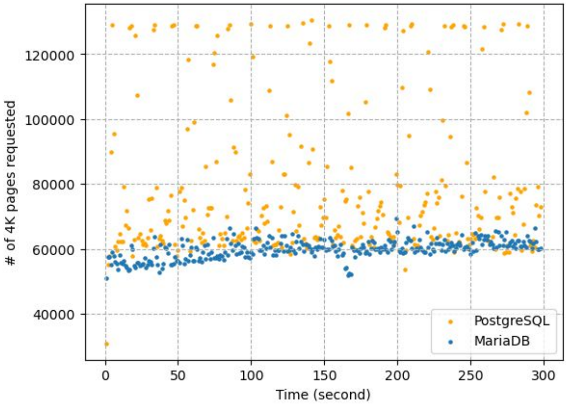
Evaluation

Evaluation Environment

- Google Compute Engine with Ubuntu 22.04.2 LTS (GNU/Linux 6.1.14 x86_64)
- Intel Xeon processor @ 2.20 GHz
- Processor has *24 cores and 128 GiB of memory*
    - 12 cores and 112 GiB of memory dedicated to NVMeVirt
    - 12 cores and 16 GiB of memory dedicated to database engine
- configured with recommended configurations from optimization tools

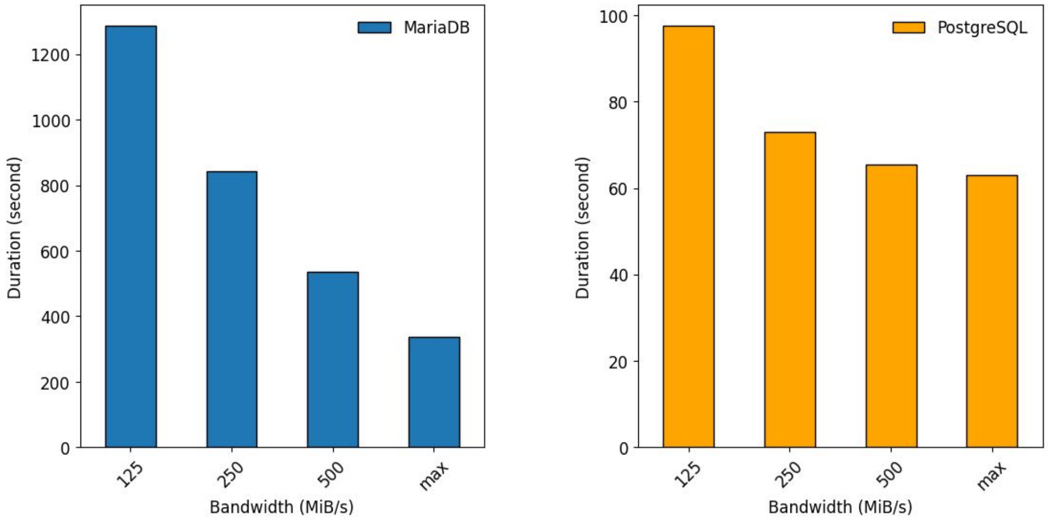
- OLTP workload with *sysbench*
- Our maximum bandwidth is around 660 MiB/s and our observations are
    1. When the bandwidth limit is low, MariaDB outperforms PostgreSQL.
    2. PostgreSQL is more sensitive to I/O bandwidth than MariaDB.
        - Performance gap becomes smaller (1.80x to 1.19x)
    3. PostgreSQL requests higher number of pages (= higher number of I/O requests).





Evaluation

- OLAP workload with *TPC-H (Query 18 - Join & Aggregation)*



- For OLAP workload, the results we observed for OLTP workload didn't appear.

➞ Hints us that *concurrency control* was what makes PostgreSQL more sensitive to I/O bandwidth for OLTP workload.

Multi-Version Concurrency Control (MVCC)

- *MVCC* is the most popular transaction management scheme in modern DBMSs.
- With *MVCC*, the DBMS maintains <u>*multiple physical versions*</u> of a single logical object in the database.
  - *Writers do <u>not</u> block readers and readers do <u>not</u> block writers.*
- *MVCC* Design Decision:
  - Concurrency Control Protocol
  - **Version Storage**
  - **Garbage Collection**
  - **Index Management**

	Protocol	**Version Storage**	**Garbage Collection**	**Indexes**
Postgres	MV-2PL/SSI	Append-Only (O2N)	Tuple-level (VAC)	Physical
MariaDB-InnoDB	MV-2PL	Delta	Tuple-level (VAC)	Logical

<MVCC Implementations>

Version Storage

- DBMS always constructs a new physical version of a tuple when a transaction updates it.
- DBMS uses the tuples' pointer field to create a *version chain* per logical tuple.

→ This allows the DBMS to find the version that is visible to particular transaction at runtime.

→ Index pointers always point to the "head" of the chain.

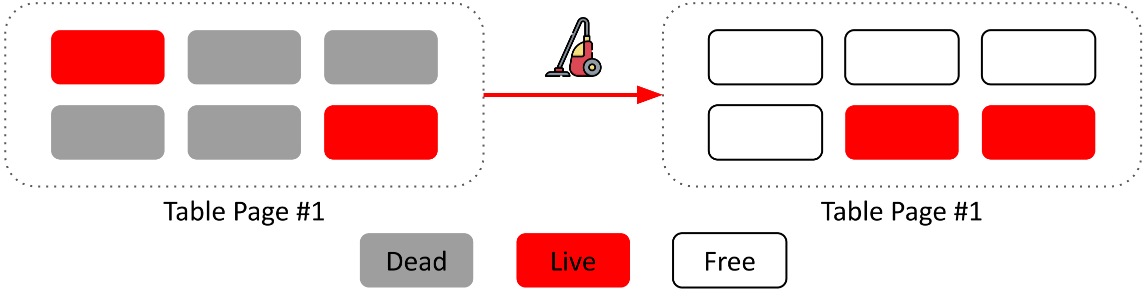
***PostgreSQL*: Append-Only Storage**

Main Table

	VALUE	POINTER
A_0	*$111*	●
A_1	*$222*	●
B_1	*$10*	Ø
A_2	*$333*	Ø

- *Copies the whole content* of the current version and *store it in the same table space*.
- Vacuum process periodically scan the table and look for reclaimable versions.

Expensive!



Version Storage

- DBMS always constructs a new physical version of a tuple when a transaction updates it.
- DBMS uses the tuples’ pointer field to create a *version chain* per logical tuple.
  - → This allows the DBMS to find the version that is visible to particular transaction at runtime.
  - → Index pointers always point to the “head” of the chain.

- *InnoDB purge thread(s) will try free UNDO log pages containing old UNDO log records.*

***MariaDB*: Delta Storage**

Main Table

	VALUE	POINTER
A_3	*$333*	●
B_1	*$10*	

Delta Storage Segment (= UNDO log)

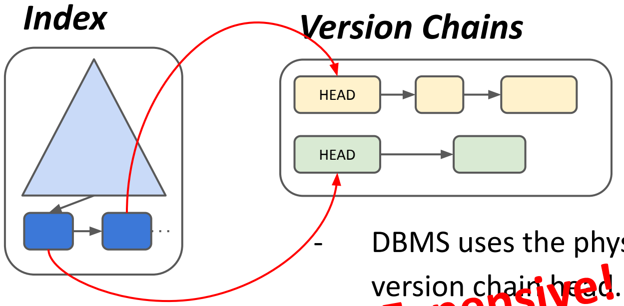
	DELTA	POINTER
A_1	*(VALUE->$111)*	Ø
A_2	*(VALUE->$222)*	●

- On every update, *copy only the values that were modified to the delta storage* and *overwrite the master version.*

Index Management

- Primary key indexes always point to version chain head.
- Using secondary index, tuple can be accessed using some piece of information other than the primary key.
  - PostgreSQL uses *physical pointers* to manage secondary indexes.
  - MariaDB uses *logical pointers* to manage secondary indexes.

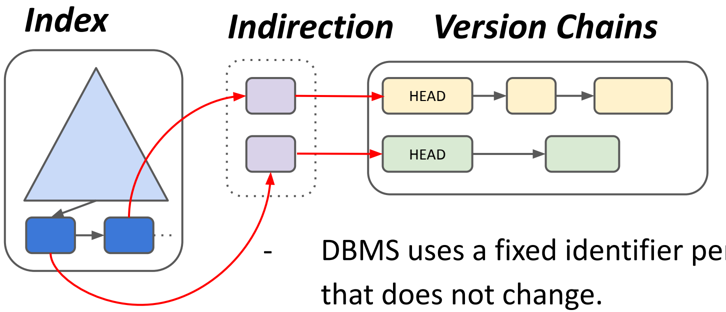
***PostgreSQL*: Physical Pointers**



- DBMS uses the physical address to the version chain head.
- **This requires updating every index when the version chain head is updated.**

Expensive!

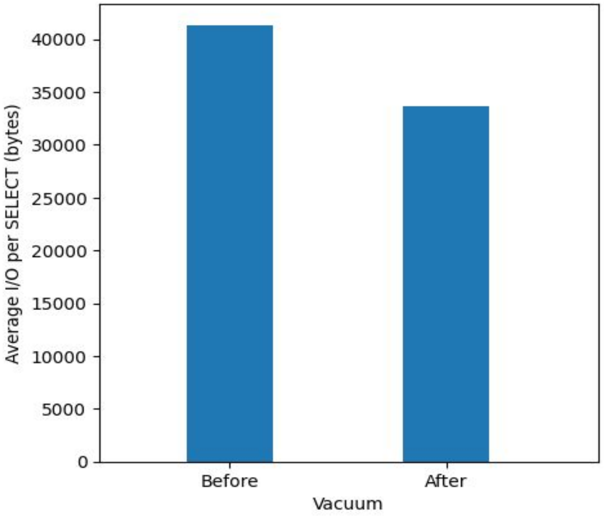
***MariaDB*: Logical Pointers**



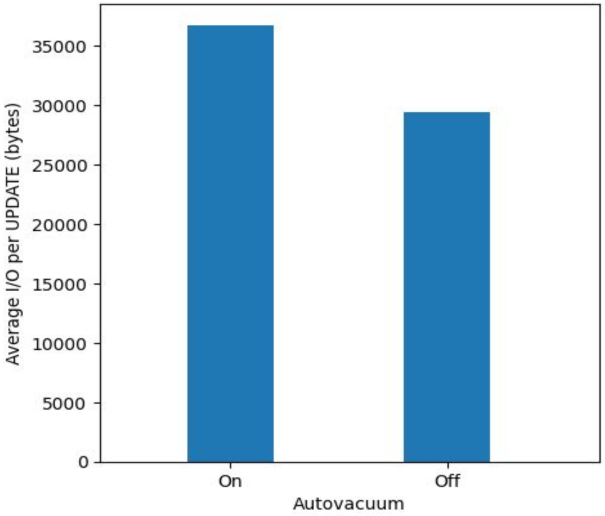
- DBMS uses a fixed identifier per tuple that does not change.
- **Updates to tuples can just update the mapping in the indirection layer.**

Experimental Analysis

- Traversing a version chain to find the latest version is requires more I/O.
- Inefficiency of *PostgreSQL*'s version storage method results in increased vacuum overhead related to I/O.



(a) Version chain scan overhead



(b) GC overhead

Conclusion

- *PostgreSQL* is designed to utilize the storage device more eagerly than *MariaDB.*
- Because of **the difference of their MVCC implementations!**
  1. Version Storage: append-only storage (*PostgreSQL*) vs. delta storage (*MariaDB*)
  2. Garbage Collection: background process vacuum (*PostgreSQL*) vs. delete undo log (*MariaDB*)
  3. Index Management: physical pointer (*PostgreSQL*) vs. logical pointer (*MariaDB*)

→ As a result, these factors result in:

- Write Amplification
- Expensive Garbage Collection
- Occupying a large amount of bandwidth