Performance Comparison&Analysis of MySQL PostgreSQL and Kunlun

Zhao Wei <twitter/linkedin/wechat: david.zhao.cn@gmail.com>

About the Author

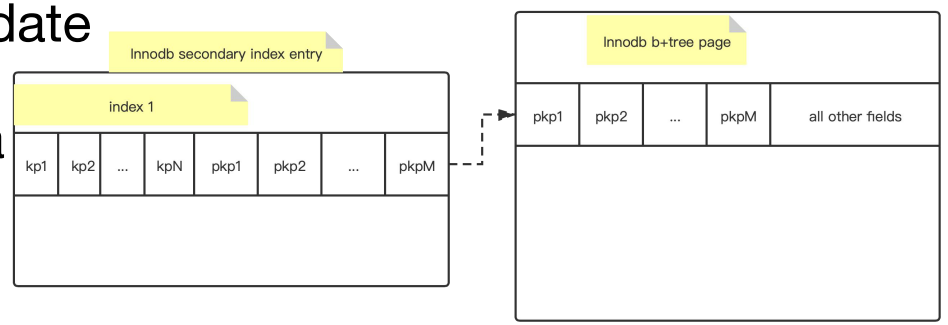
- Zhao Wei (David Zhao) twitter/linkedin/wechat: david.zhao.cn@gmail.com
- Database kernel developer in Oracle & Tencent for over a decade
- Kunlun Distributed DBMS
  - Goals&objectives
    - A NewSQL distributed OLTP DBMS with industry level robustness
    - Premium scalability, availability, fault-tolerance&crash safety, and reliability
    - Premium performance, ease of use&administration, and autonomousity;
    - Cloud native & DBaaS on public clouds
  - Finished all DDL & DML functionality, Kunlun DBMS is ready for POC and evaluations
  - All modules are open source at https://github.com/zettadb/
- Founded ZettaDB in late 2020
  - with a top notch team of fomer database kernel developers from Tencent, Huawei, Alibaba, etc
  - Visit www.zettadb.com for more info, documentation, resources and downloadable binary programs&docker images and more.

Agenda

- Comparisons of DB Kernel Mechanisms and Performance Implications
  - Storage Structures
  - Undo Logging
  - Concurrency Control
  - Architectures & More
- PostgreSQL vs. Kunlun-storage Performance Comparisons & Analysis
- Kunlun Architecture, Advantages & Highlights
- Kunlun-storage performance enhancements & Performance Comparisons with Percona-mysql-8.0.22

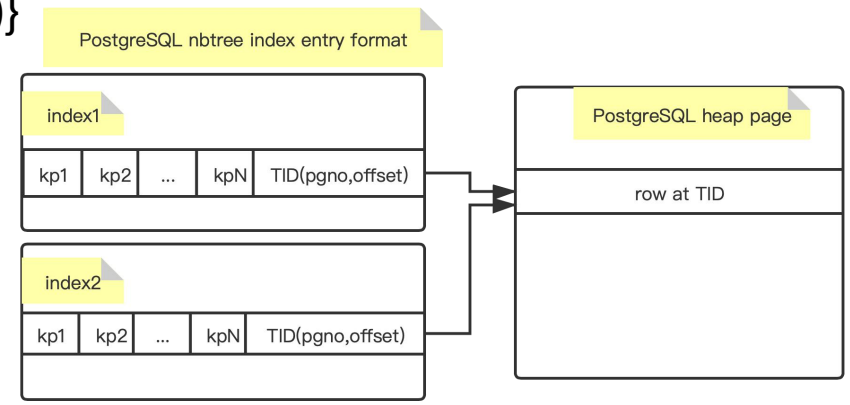
Storage Structures

- InnoDB: b+ tree
  - Table: PK as key, or rowid as key, clustered index
    - maintain PK index order at each row insert/PK update
    - High PK range locality
    - update: latest version on page, undo logging delta
    - delete: set 'deleted' mark, purge later asyncly
  - secondary index(sec-idx):
    - {header, key fields, remaining PK fields}
    - maintain sec-idx key order at each insert/key update
      - change buffering improves performance
    - row updated: insert new key entry only if key field(s) changes
    - row deleted: mark key entry deleted, purge later asyncly, changes buffered
  - access by secondary index
    - Step 1: in sec-idx, sec-idx key->PK => Step 2: in primary table, PK->row
    - range scan: sec-idx key range scan->accumulate PKs=>{ [sort PKs => ordered multi-range-read by PK in primary table] / [out of order row reads one by one by PK] }



Storage Structures

- PostgreSQL: heap + nbtree
  - Table: heap
    - tuples unordered, or insert order by default
    - fast bulk load, esp. no secondary indexes
    - update: leave old tuple as is, insert full tuple with updated fields, chain up with prev version,insert index key entries for all indexes
    - delete: mark tuple deleted, leave sec-idx key entries as is
  - primary or secondary index: nbtree, one file for one nbtree/index
    - {key fields, TID(tuple ID, or tuple physical position(pageno, offset))}
    - Low PK range locality
    - TID reference: position dependence, pinned inside a page
      - One index entry for every tuple
        - Index bloat and grouped TIDs optimization
        - extra page accesses
      - HOT updates: new tuple in same page & no indexed field updated
  - access by secondary index
    - in sec-idx, sec-idx key->TID => in primary table, fetch row at TID
    - range scan: sec-idx->TIDs => tidbitmapscan in heap
    - include fields: covering index



Undo Logging

- InnoDB: dedicated undo log tablespace
  - each txn: 1 insert undo segment if does insert and/or 1 update undo-seg
  - insert undo seg content: PK of newly inserted rows, table-id
  - update undo seg content: old version of updated&deleted rows' fields + all above
  - usage 1: rollback a txn, undo page changes
  - usage 2: build old version rows for MVCC
  - only latest tuple version stored in table file
    - older versions can be reconstructed following the undo log chain of the target row
    - one secondary index entry for each row

Undo Logging

- PostgreSQL: no undo log segment
  - Full old tuples for each row chained(via TID) & scattered in heap pages
    - table file bloat for heavily updated tables
  - One index entry for each heap tuple in every sec-idx except rare HOT updates
    - index file bloat
  - rollback: leaves dead tuples as is in-place
  - MVCC read/write: follow version chain to fetch each row's visible tuple
    - random & excessive page accesses
  - space reclamation&cleanup: vacuum
    - massive buffer page evictions
      - massive following page load & evictions like a series of shock waves
    - massive IO bandwidth consumption
    - 32bit txn id wrapping issue makes vacuuming inevitable even without updates/deletes

Concurrency Control

- InnoDB
  - MVCC for select stmts
  - row/gap txnal locks for insert/delete/update stmts
    - conficiting writes will all get a chance to execute and complete unless in a deadlock
      - no resource waste/jitters from txn aborts
    - “what you see isn't what you update/delete”
      - select * from orders where userid > 123
      - update orders set discount=discount*1.1 where userid > 123;
      - the significance for “ update/delete...returning... “
        - kunlun-storage feature
    - Concurrency impacted when non-target rows scanned&locked in RR isolevel
      - for updates&deletes

Concurrency Control

- PostgreSQL
  - MVCC for reads & writes: strengths&weakness
    - conflicting writes cause problems: Abort all conflicting txns
      - isolation_level = Repeatable Read
      - concurrent txns T1&T2 both updates row R, T1 updates row R first
      - T2: block&wait for T to end
        - T1 commit (likely) => T2 aborts;
        - T1 aborts (unlikely) => T2 proceed
      - big waste of computing resources in heavily contented workloads
      - longer average latency
      - Jittering latency and throughput
    - No row/page level locks for write workloads: trivial lock system memory/time overhead
      - Perfect for uncontented write workloads: append only
      - But storage format is row oriented rather than column oriented, not perfect for timeseries workloads

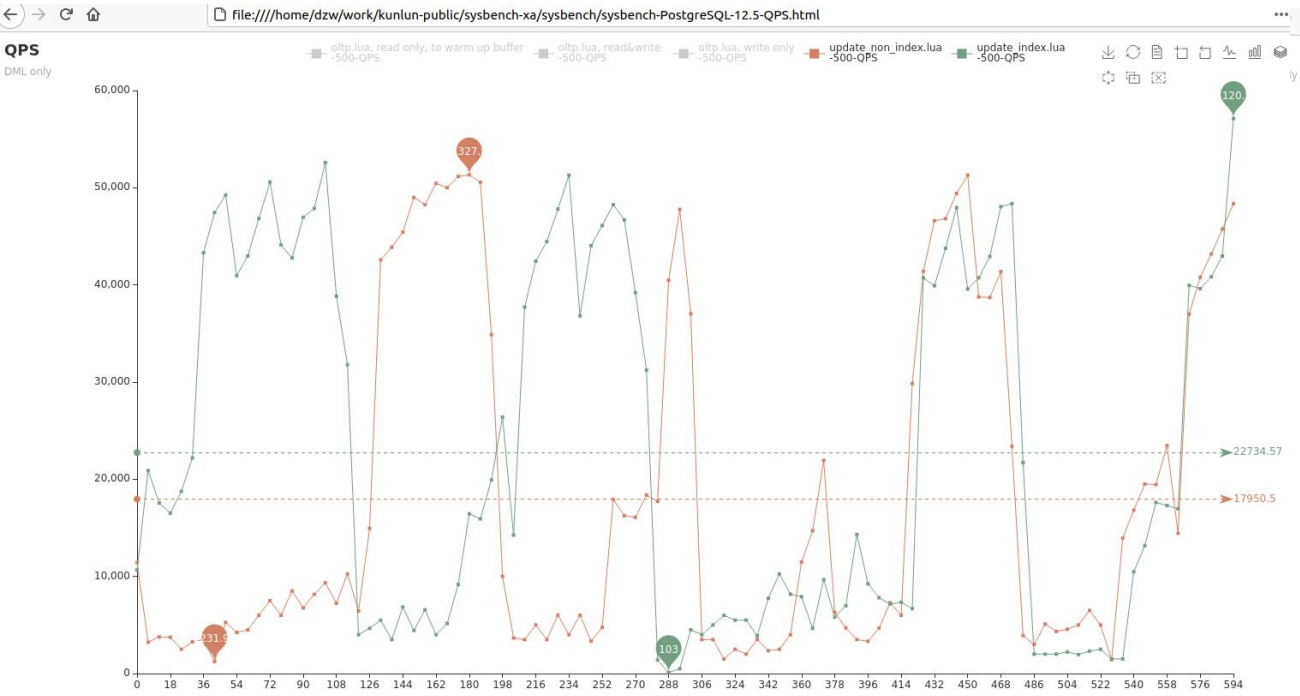
Architectures & More

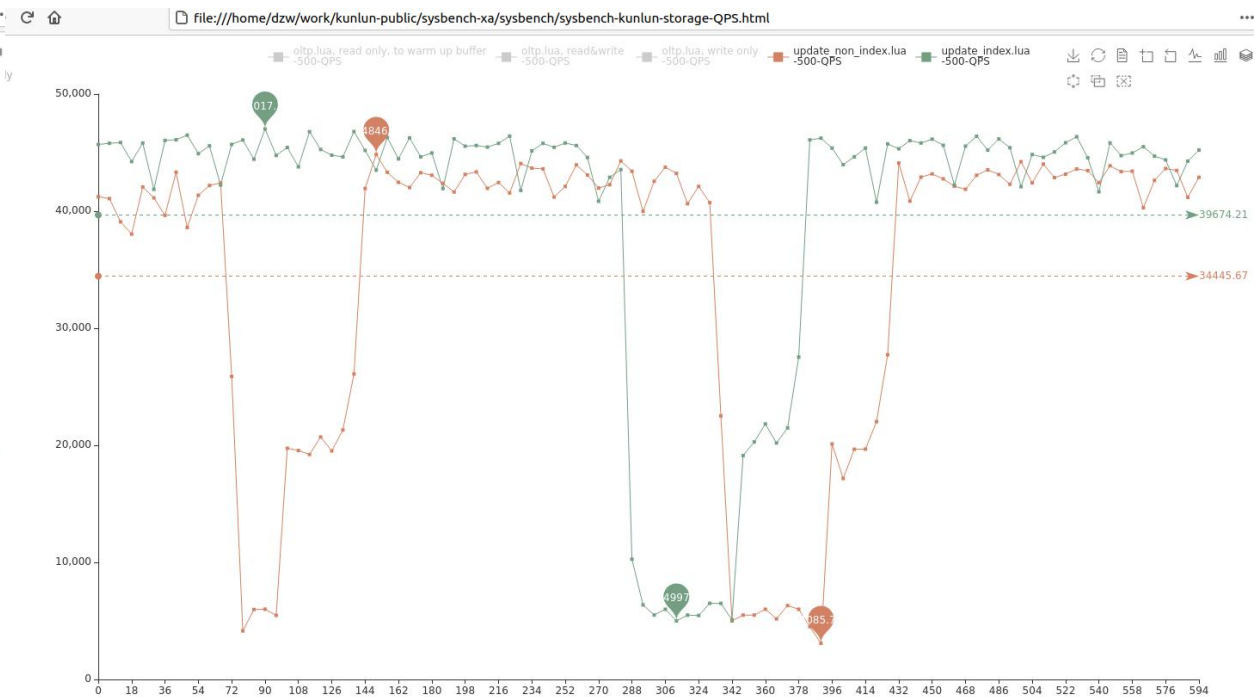
- PostgreSQL: multi-process, one process for each connection
  - Duplicated kernel data structures in every backend process
    - page table: references to buffer pool physical page frames
      - hugetlb may not be used even if huge_pages=try
    - Massive kernel memory consumption
      - Example: 256GB buffer pool, 4KB page frame, 8 bytes for each PTE, no hugetlb in effect
        - Each PostgreSQL backend proc's page table can be at most 0.5GB!
        - Not many concurrent connections possible!
      - Huge costs for context switches
    - Limited client connections, often < 500, seldom above 1000.
      - limited throughput and concurrency
- None of above issues exist in MySQL with thread pool
  - One process, multi-threads in thread pool
    - Always only one page table for one mysql instance no matter how many client connections
    - thread context switches much cheaper than process!
    - massive concurrent connections(10K or more) can be by handled by hundreds of threads
      - common for big IT systems serving mobile apps

Agenda

- Comparisons of DB Kernel Mechanisms and Performance Implications
  - Storage Structures
  - Undo Logging
  - Concurrency Control
  - Architectures & More
- PostgreSQL vs. Kunlun-storage Performance Comparisons & Analysis
- Kunlun Architecture, Advantages & Highlights
- Kunlun-storage performance enhancements & Performance Comparisons with Percona-mysql-8.0.22

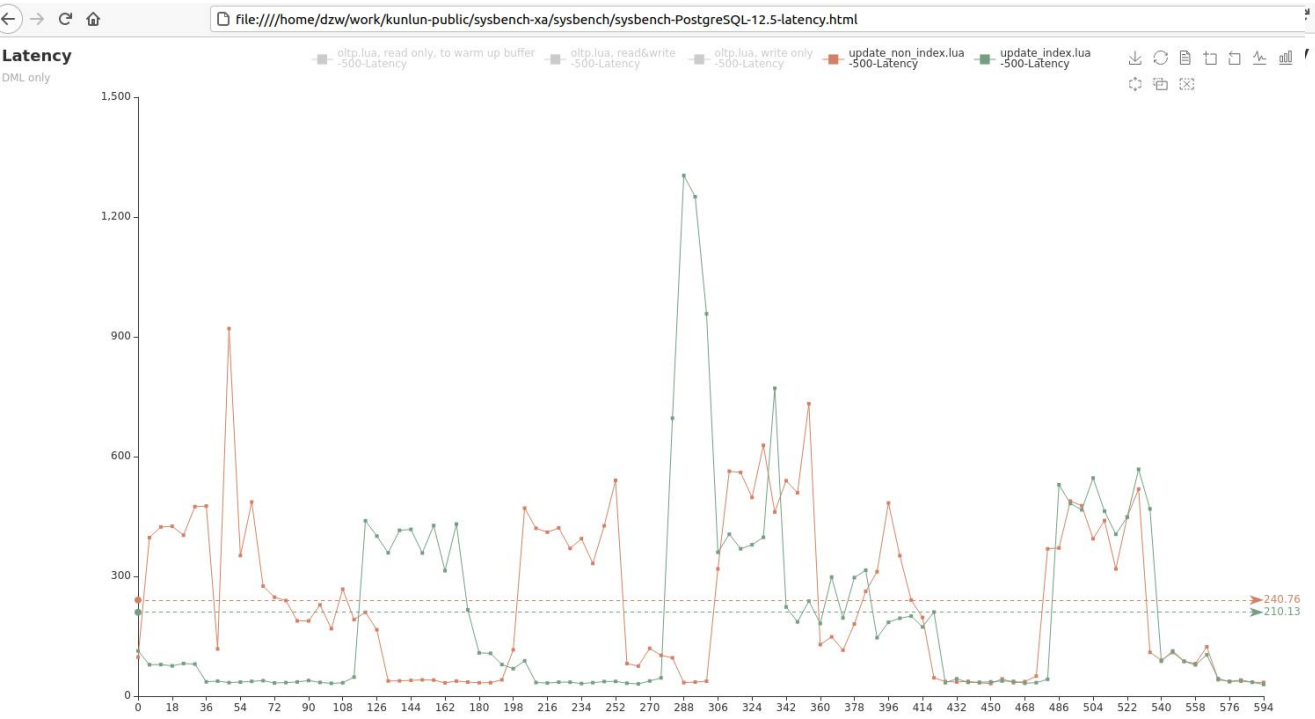
update_[non_]index QPS

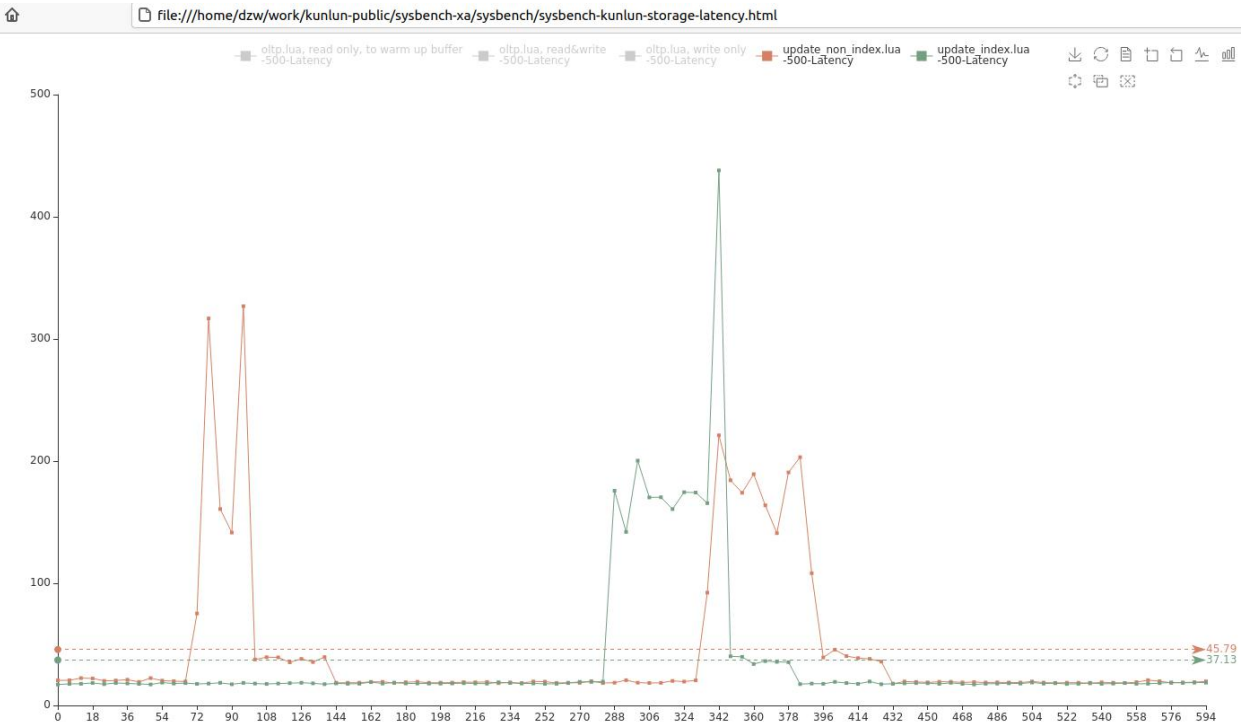




Product	case	95% latency(ms)	avg-latency(ms)	latency-stddev(ms)	QPS	QPS-stddev
kunlun-storage	update_non_index	43	16	73	30357	15459
PostgreSQL-12.5	update_non_index	79	25	180	19740	16926
kunlun-storage	update_index	35	15	71	32843	15220
PostgreSQL-12.5	update_index	80	26	363	18971	17658

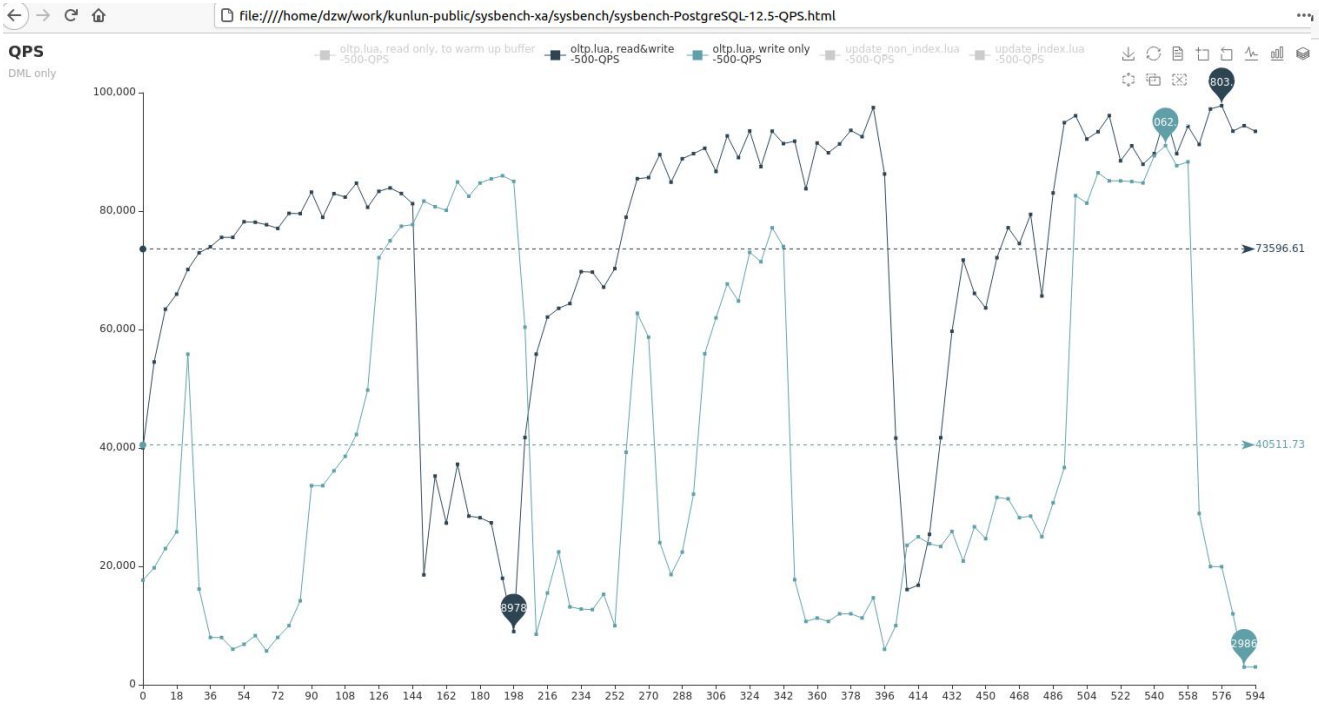
update_[non_]index latency

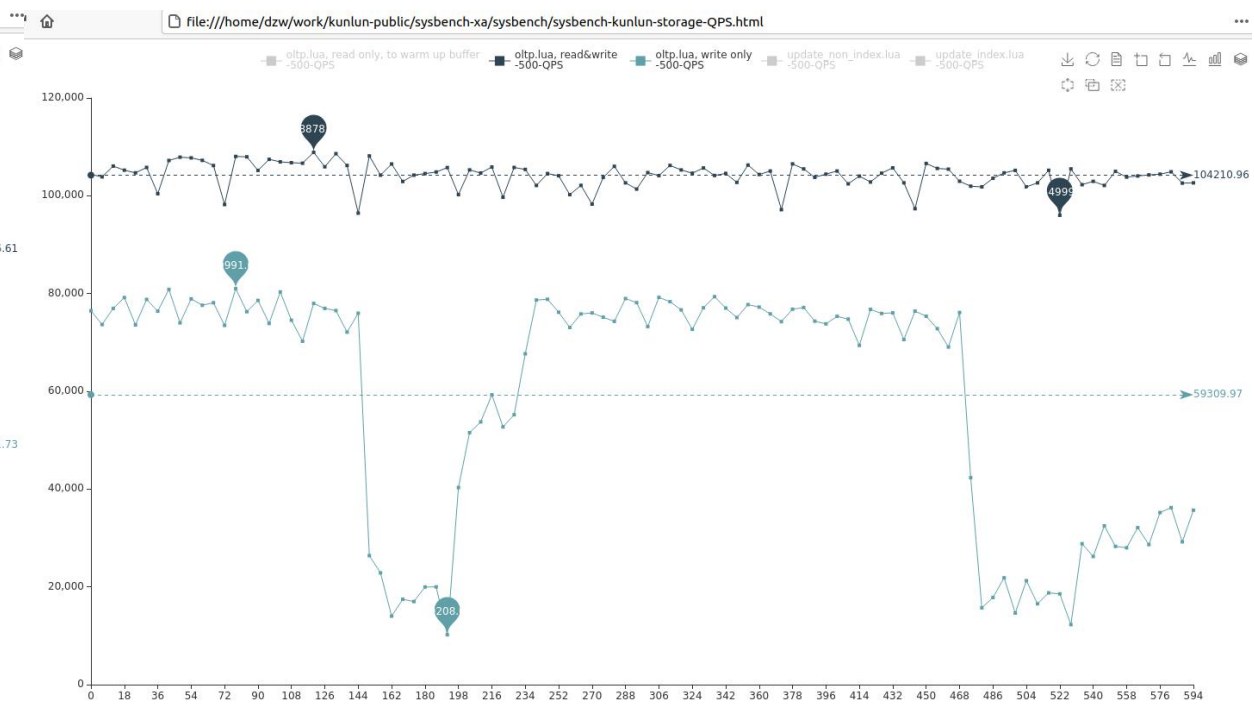




Product	case	95% latency(ms)	avg-latency(ms)	latency-stddev(ms)	QPS	QPS-stddev
kunlun-storage	update_non_index	43	16	73	30357	15459
PostgreSQL-12.5	update_non_index	79	25	180	19740	16926
kunlun-storage	update_index	35	15	71	32843	15220
PostgreSQL-12.5	update_index	80	26	363	18971	17658

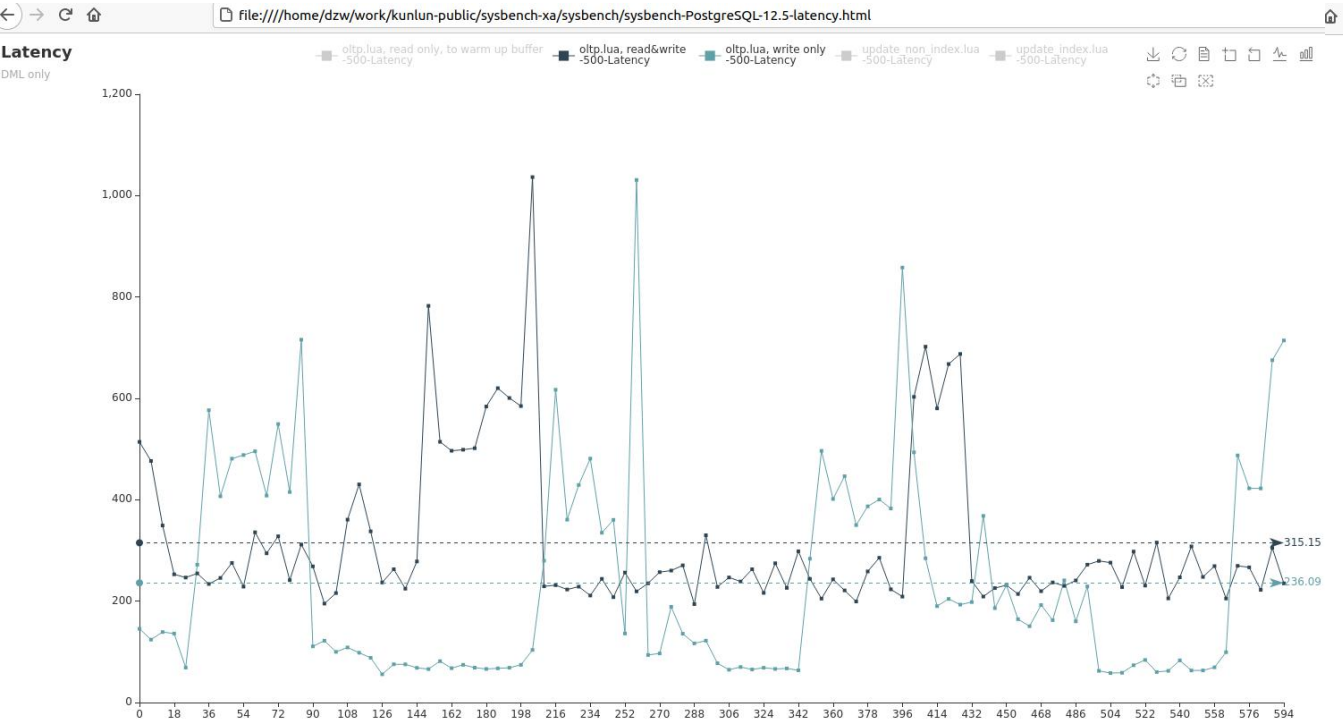
oltp w/rw QPS

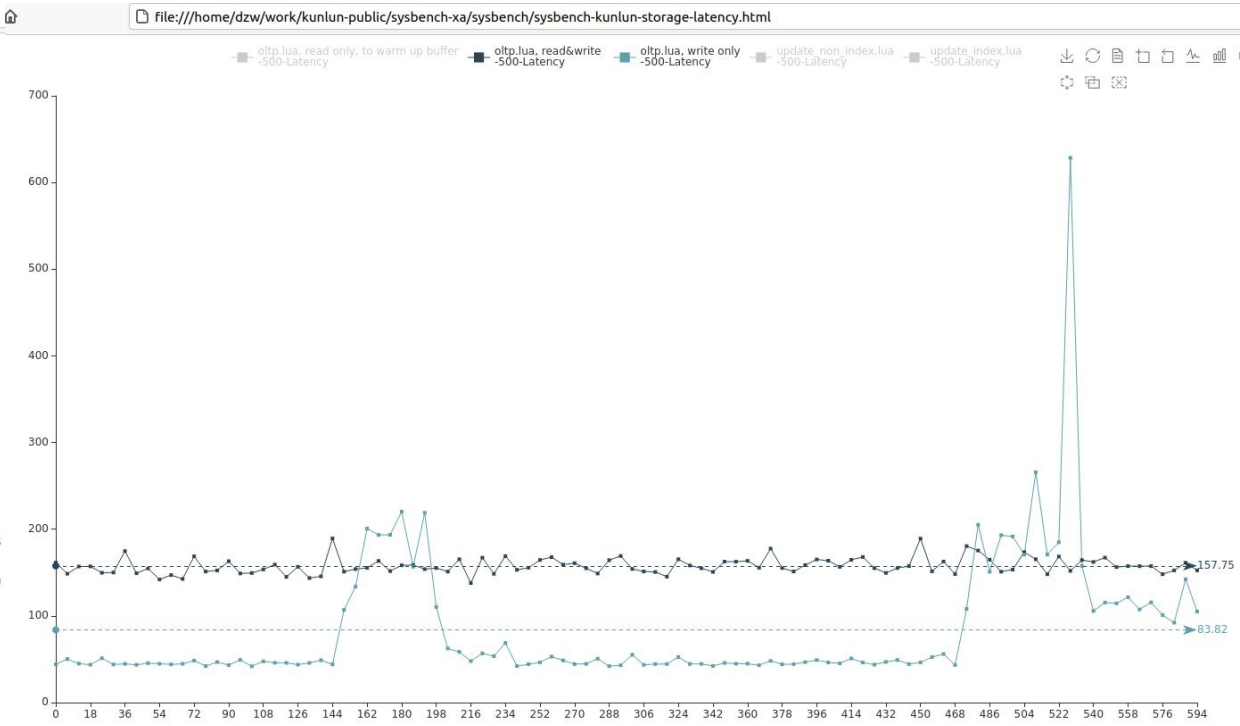




Product	case	95% latency(ms)	avg-latency(ms)	latency-stddev(ms)	QPS	QPS-stddev
kunlun-storage	oltp write	80	34	83	58670	22751
PostgreSQL-12.5	oltp write	130	47	196	42248	29791
kunlun-storage	oltp read/write	167	91	47	98603	12885
PostgreSQL-12.5	oltp read/write	264	108	135	83280	24053

oltp w/rw latency





Product	case	95% latency(ms)	avg-latency(ms)	latency-stddev(ms)	QPS	QPS-stddev
kunlun-storage	oltp write	80	34	83	58670	22751
PostgreSQL-12.5	oltp write	130	47	196	42248	29791
kunlun-storage	oltp read/write	167	91	47	98603	12885
PostgreSQL-12.5	oltp read/write	264	108	135	83280	24053

Kunlun-storage VS PostgreSQL Performance Comparison Conclusions

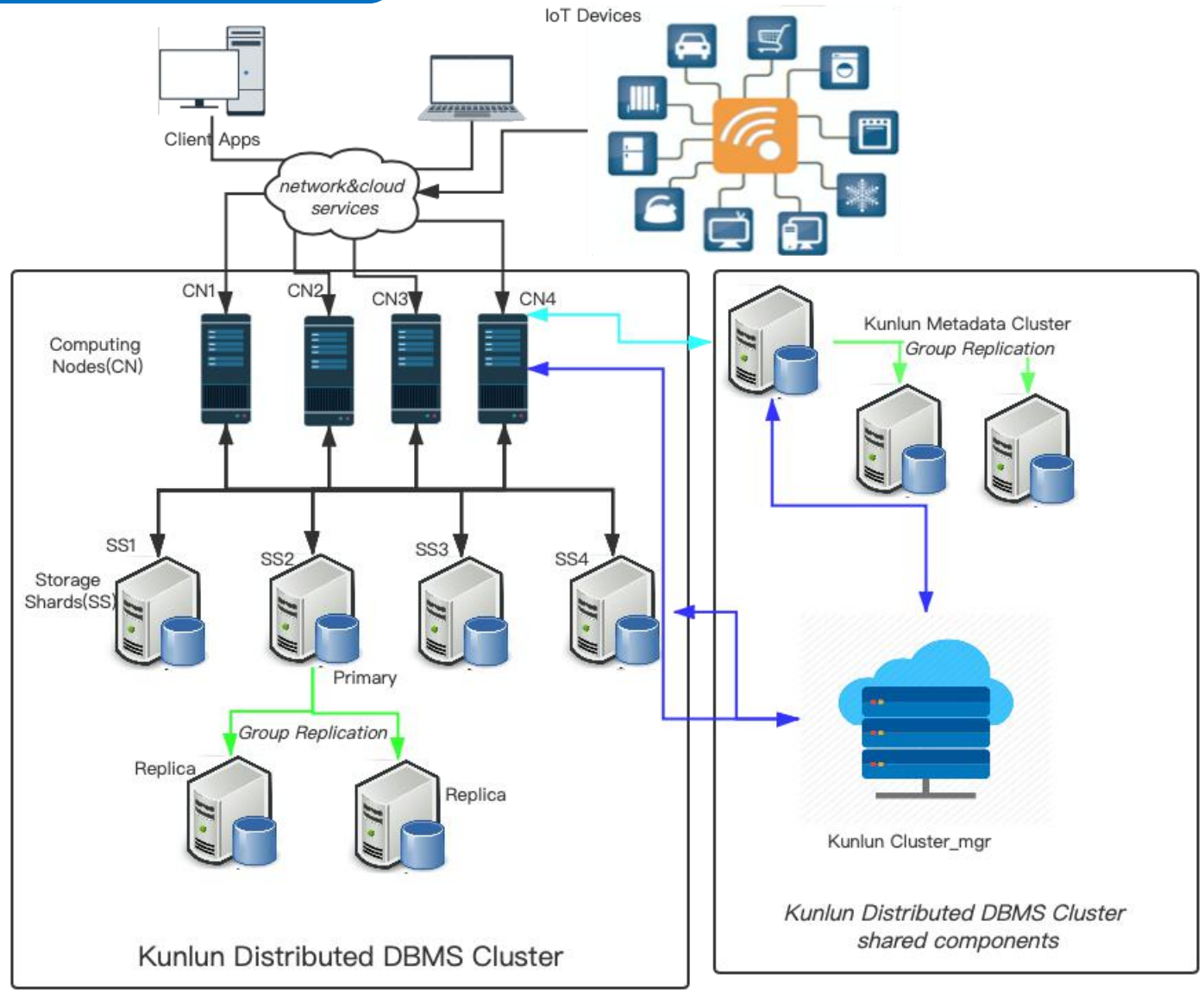
- kunlun-storage has much higher QPS(50%+) than PostgreSQL in update_[non_]index&oltp write test cases
  - caused by PostgreSQL's storage engine issues
- kunlun-storage has much lower latency(50%~65%) than PostgreSQL in all test cases
- kunlun-storage oltp.lua read&write case QPS 20% higher than PostgreSQL
  - PostgreSQL read-only query processing is as good as mysql or better
- kunlun-storage has much smaller latency stddev(20%~40%) in all test cases than PostgreSQL
  - PostgreSQL latency isn't stable/smooth, random&frequent slow-downs
  - client timeouts could happen occasionally or often under heavy workloads with PostgreSQL
- kunlun-storage QPS&latency curve appears much more smooth than PostgreSQL
  - IT systems using kunlun DBMS will bring better end user experience
- Parallel data load time: kunlun-storage took twice the time of PostgreSQL

Agenda

- Comparisons of DB Kernel Mechanisms and Performance Implications
  - Storage Structures
  - Undo Logging
  - Concurrency Control
  - Architectures & More
- PostgreSQL vs. Kunlun-storage Performance Comparisons & Analysis
- Kunlun Architecture, Advantages & Highlights
- Kunlun-storage performance enhancements & Performance Comparisons with Percona-mysql-8.0.22

Kunlun Architecture

- Combine the best of MySQL&PostgreSQL
  - PostgreSQL: Query processing
    - Premium query processing performance
    - Aggregates&Window Functions
    - Stored Procedures
    - Open architecture
  - MySQL: Innodb & Binlog Replication
    - Premium performance in innodb for heavy workloads
      - write intensive & highly concurrent
      - massive connections
    - Flexible HA via binlog
    - Open ecosystem using binlog



Kunlun Advantages & highlights

- Share nothing for scalability
  - distribute tables/table partitions to multiple storage shards
  - multiple nodes for read & write queries
- Distributed transaction processing & distributed query processing
  - transparently available, need no extra work for users
  - read/write data sharded across multiple storage shards the same way with a standalone db
- Elastical scalability
  - Automatic & continuous & on demand
  - unnoticible by application or end users
- Crash safety, fault tolerance, high availability
  - any node/network fault at any time won't break transaction ACID guarantees
  - Computing nodes auto failover for storage shards
- All PostgreSQL connectivity
  - JDBC/ODBC & all language bindings
- MySQL client protocol and private DML grammar (pending)
  - All language bindings for MySQL

Kunlun Advantages & highlights

- Support most PostgreSQL query processing features
  - Most DDLs and DMLs grammars
  - tables(partitioned or standalone), indexes, views, materialized views, sequences
  - Most PostgreSQL query features across multiple storage shards, including:
    - subquery, join, aggregations, window functions, CTE, prepared statement, insert/delete/update... returning
    - stored procedures
  - All basic data types(numeric types, string/text/blob types, date/time/timestamp, enum, etc)
  - Unsupported features: foreign keys, triggers, event triggers, type inheritence, composite types, tablespaces, etc
- Light weight computing node backend process: no duplicate kernel data structures
  - No shared buffer pools => no huge page table in each backend
  - No local table files => no massive open FDs
  - Small memory footprint for each backend
  - cheap context switching
- Computing node initializes from metadata server
  - Has synchronized local states
  - No need for HA measures: kunlun DBMS cluster daily DBA work is mostly for MySQL instances

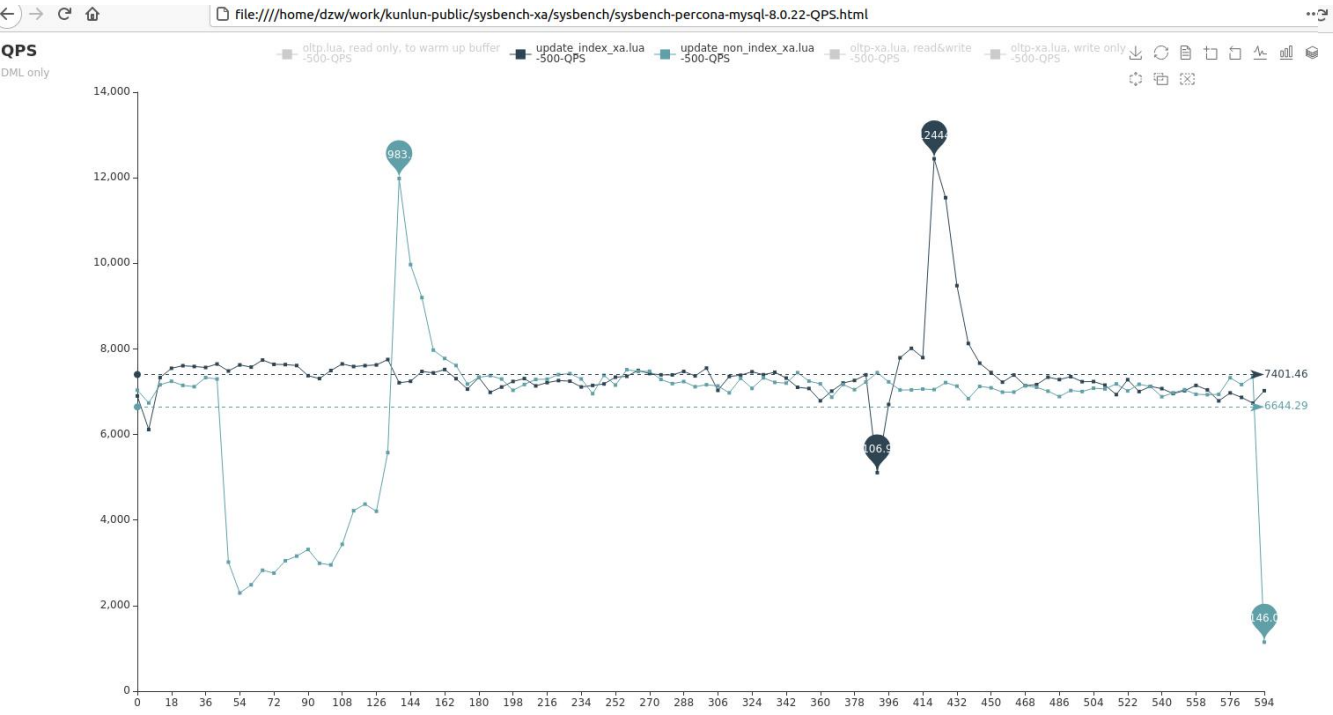
Agenda

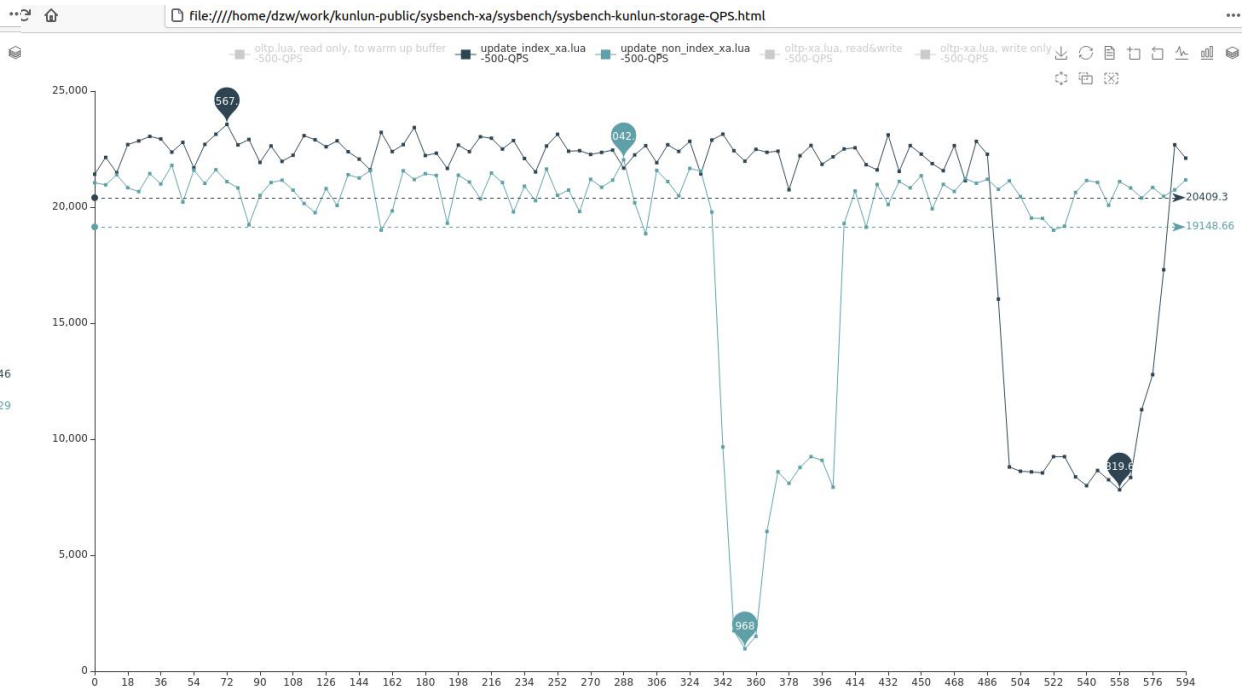
- Comparisons of DB Kernel Mechanisms and Performance Implications
  - Storage Structures
  - Undo Logging
  - Concurrency Control
  - Architectures & More
- PostgreSQL vs. Kunlun-storage Performance Comparisons & Analysis
- Kunlun Architecture, Advantages & Highlights
- Kunlun-storage performance enhancements & Performance Comparisons with Percona-mysql-8.0.22

Kunlun-storage Performance Enhancement

- Accelerating XA transaction commit performance by up to 200% to 300% than percona-mysql-8.0.22
  - An 2PC txn contains 2 event group in binlog, each with unique gtid
  - Official MySQL-8.0 clone mechanisms
    - gtid -> undo log at regular txn commit and XA txn prepare/rollback/commit
    - gtid_persistor: asyncly&periodically flush gtid@undo log header to mysql.gtid_executed
    - XA COMMIT
      - block&wait for gtid@undo log header to be flushed after waking up gtid_persistor
      - waken up by gtid_persistor
      - store 2nd gtid to the same gtid slot in undo log header
      - huge time&system resource cost from waits & context switching!
  - Kunlun-storage enhancement(open source at github.com/zettadb/kunlun-storage)
    - Alloc 2 gtid slots in update undo segment, one jslot for each event group
    - no wait for gtid_persistor at XA COMMIT
    - XA txn performance soared by over 200% percent
    - uncompatible with data directories of official mysql-8.0
    - works well with official percona xtrabackup
    - happy to contribute it to MySQL and/or Percona team(s)

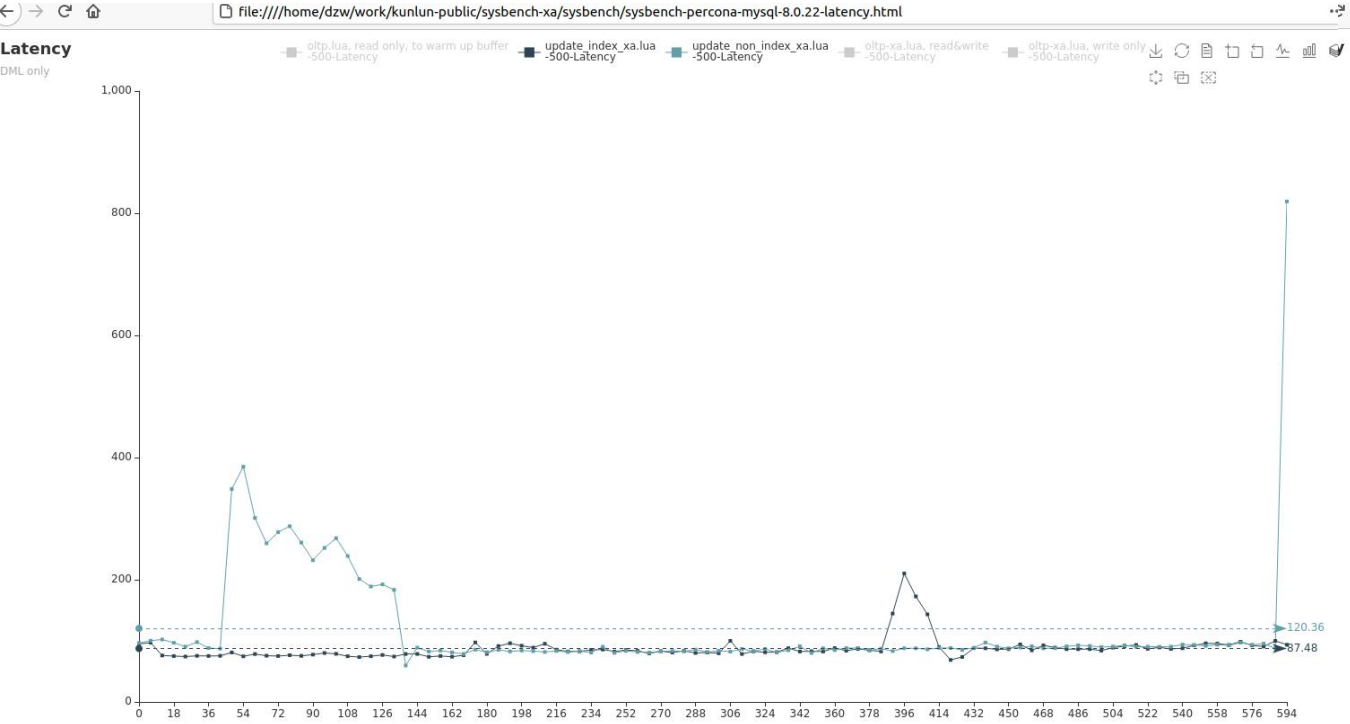
update_[non_]index_xa QPS

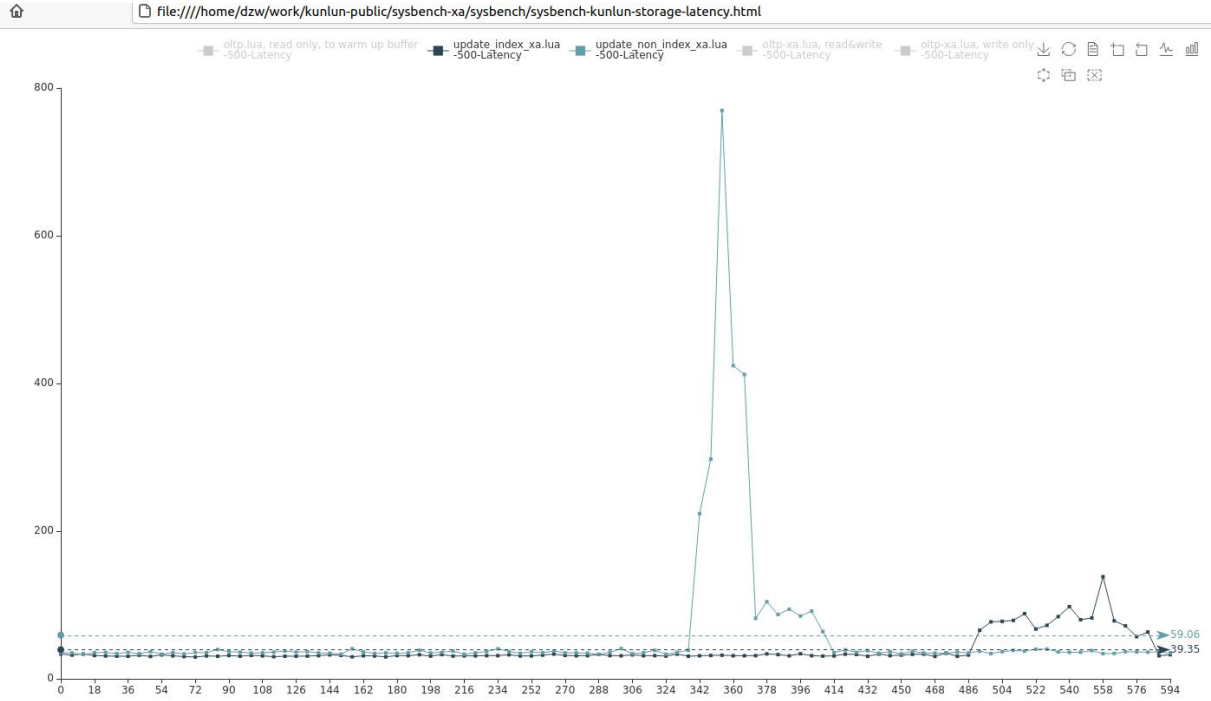




Product	case	95% latency(ms)	avg-latency(ms)	latency-stddev(ms)	QPS	QPS-stdev
kunlun-storage	update_non_index_xa	52	28	107	17553	6821
percona-mysql	update_non_index_xa	160	80	195	6195	2337
kunlun-storage	update_index_xa	46	27	105	18624	6834
percona-mysql	update_index_xa	146	79	208	6294	2374

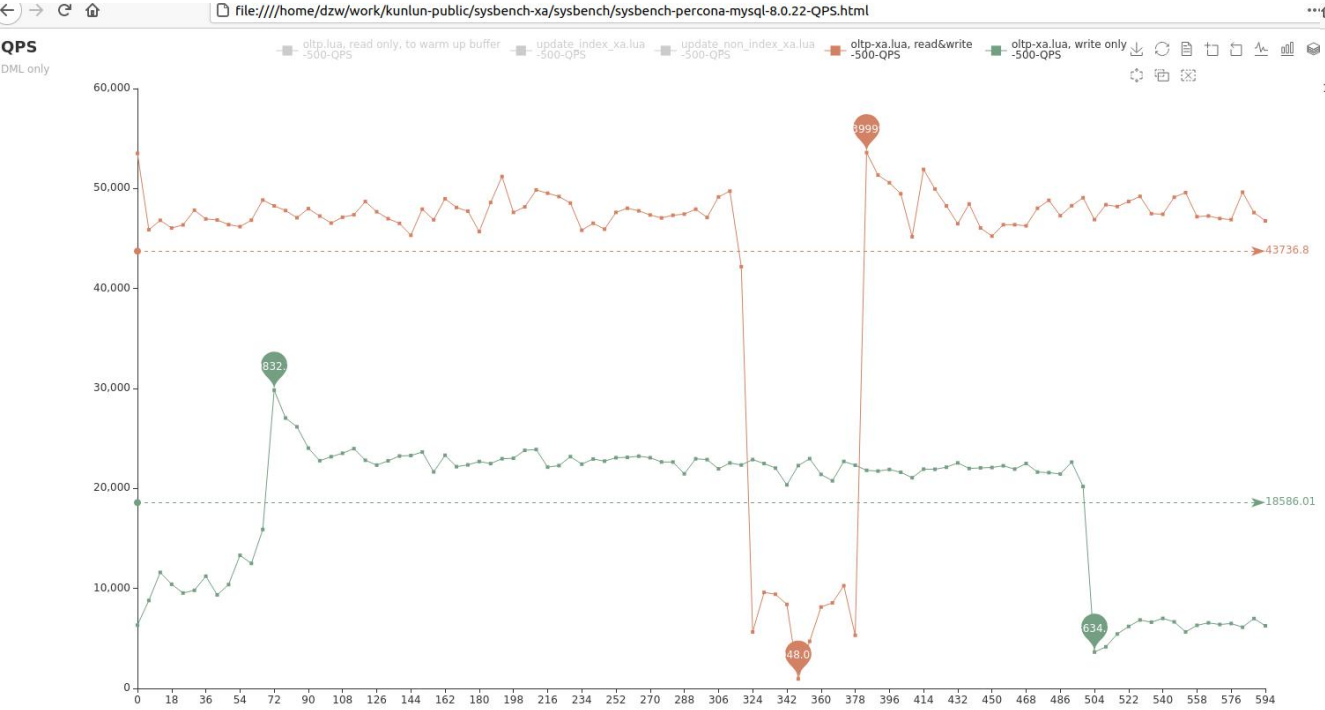
update_[non_]index_xa latency

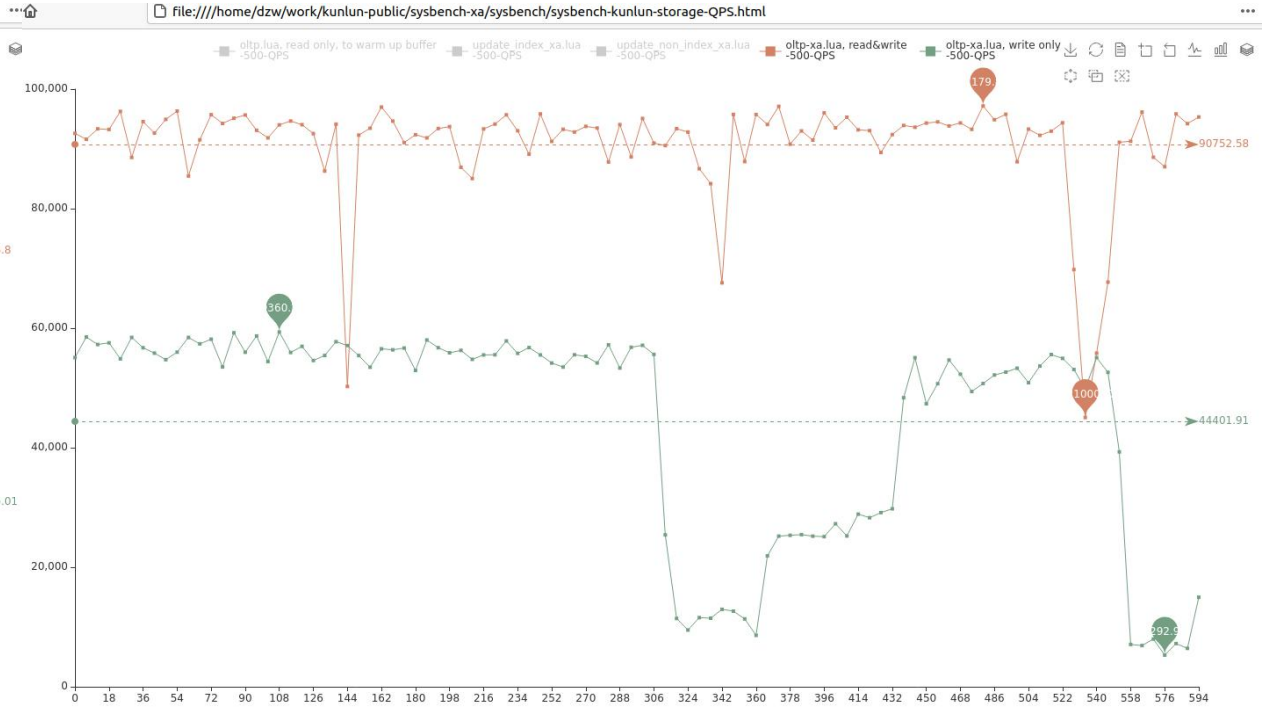




Product	case	95% latency(ms)	avg-latency(ms)	latency-stddev(ms)	QPS	QPS-stdev
kunlun-storage	update_non_index_xa	52	28	107	17553	6821
percona-mysql	update_non_index_xa	160	80	195	6195	2337
kunlun-storage	update_index_xa	46	27	105	18624	6834
percona-mysql	update_index_xa	146	79	208	6294	2374

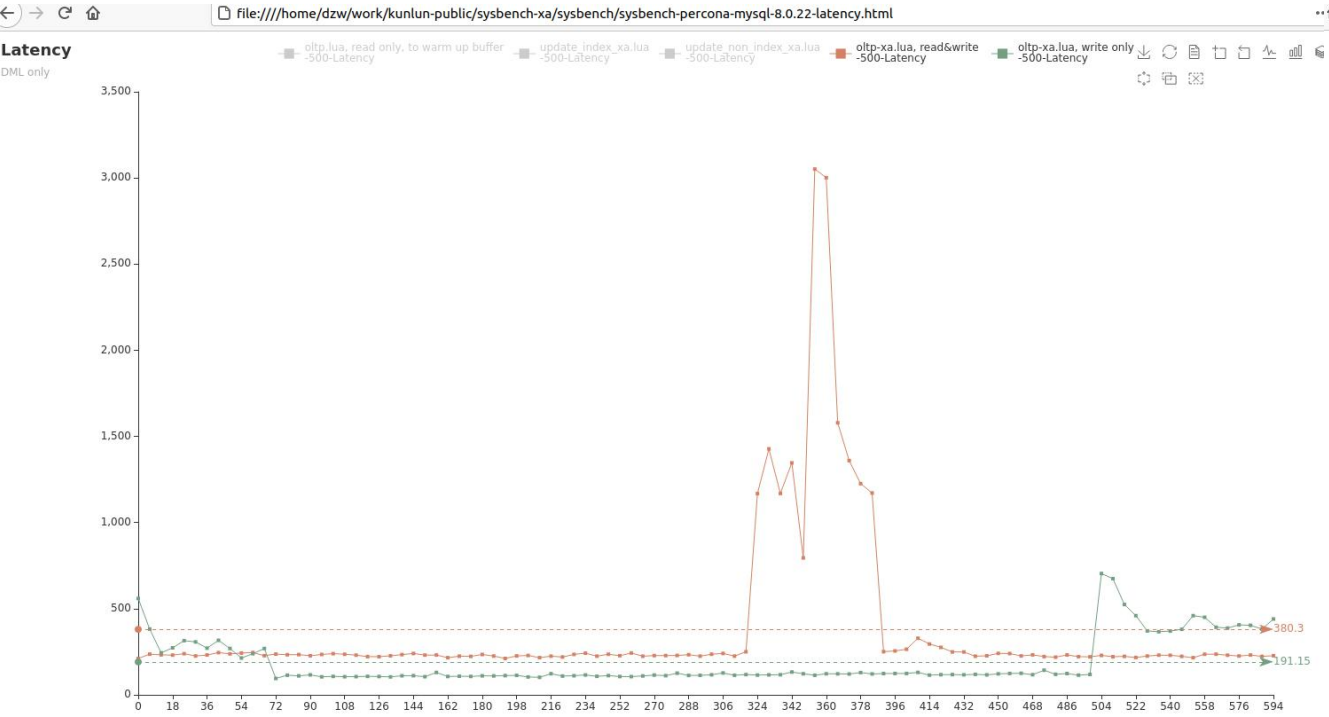
oltp_xa w/rw QPS

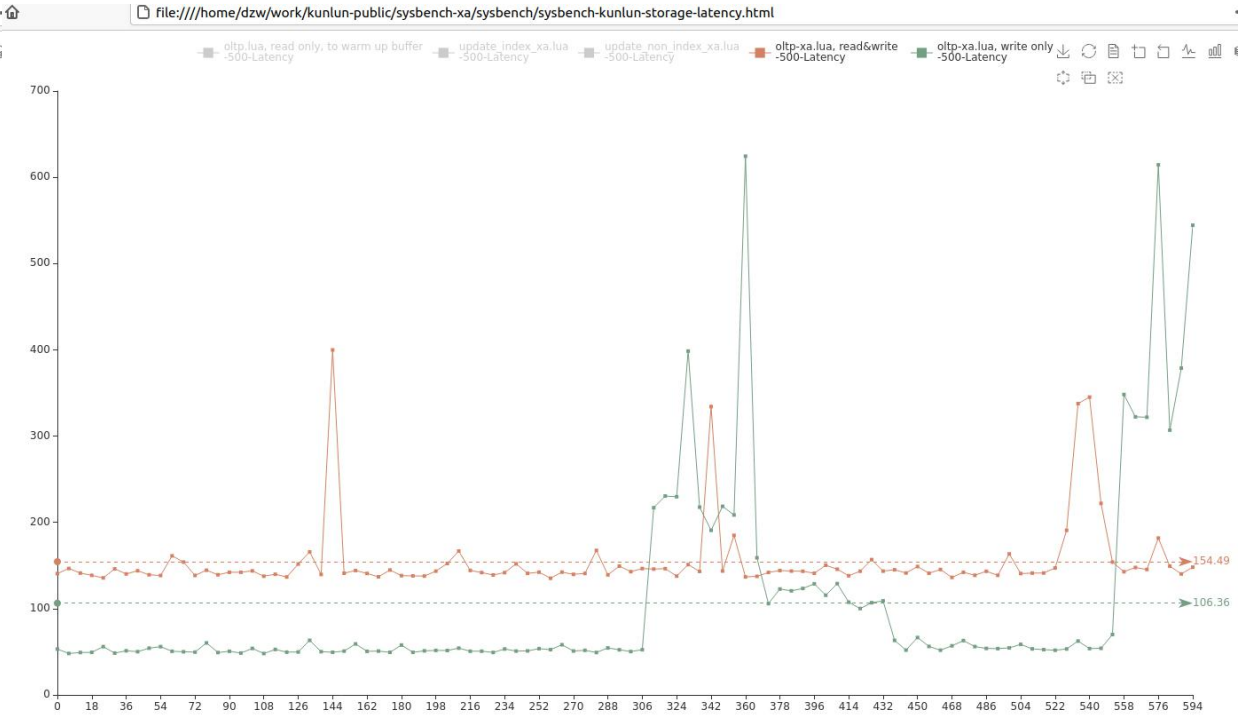




Product	case	95% latency(ms)	avg-latency(ms)	latency-stddev(ms)	QPS	QPS-stdev
kunlun-storage	oltp_xa write	90	45	119	43960	18030
percona-mysql	oltp_xa write	226	107	218	18664	7011
kunlun-storage	oltp_xa read/write	158	100	64	89893	13991
percona-mysql	oltp_xa read/write	313	210	276	42853	10495

oltp_xa w/rw latency





Product	case	95% latency(ms)	avg-latency(ms)	latency-stddev(ms)	QPS	QPS-stdev
kunlun-storage	oltp_xa write	90	45	119	43960	18030
percona-mysql	oltp_xa write	226	107	218	18664	7011
kunlun-storage	oltp_xa read/write	158	100	64	89893	13991
percona-mysql	oltp_xa read/write	313	210	276	42853	10495

Kunlun-storage VS MySQL Performance Comparison Conclusions

- MySQL: Using Percona-server-8.0.22
  - performance issues we fixed exist in official MySQL code
- In update_[non_]index_xa.lua cases, kunlun-storage shows 3 times better performance than mysql
  - latency: kunlun-storage is 1/3 that of mysql and half stddev
    - much better end user experience: smooth and quick
  - QPS: kunlun-storage is 3 times that of mysql
- In oltp_xa.lua write/read-write cases, kunlun-storage shows 2 times better performance than mysql
  - latency: kunlun-storage is 1/2 that of mysql and half stddev
    - much better end user experience: smooth and quick
  - QPS: kunlun-storage is 2 times that of mysql
- kunlun-storage shows 2-3 times better performance than community mysql or percona-server

Kunlun DBMS vs. Kunlun-storage Performance Comparison & Conclusions

- Kunlun DBMS cluster setup
  - 2 storage shards running on 2 computing servers S1&S2
  - 1 computing node running on S1, sysbench running on S2
- For each SQL stmt from client, extra time cost:
  - computing node query processing
    - parse, resolve, optimization, and local execution for selects
    - more performance enhancements underway
  - computing node<--->storage shard primary transfer
    - ignorable (0.1ms, 0.5ms) if on same rack
  - latency growth is limited&confined
- update_[non_]index.lua cases
  - 1 SQL stmt writing to 1 table, no 2PC
  - 95% latency grew by 6~8ms or 20%, QPS down by 20%
- oltp.lua write only
  - 4 SQL stmts writing to the same table, no 2PC
  - 95% latency grew by 25ms or 40%, QPS down by 15%
- oltp-randtabs.lua write only
  - 4 SQL stmts, writing to random tables in each stmt, insert&delete the same table, 75% 2PC, 25% 1PC
  - 95% latency grew by 50ms or 100%, QPS down by 50%
  - commit log group write & 2nd phase causes 25ms on average for a txn
- Kunlun will out perform when a single kunlun-storage shard has to scale out
  - hardware resources/network bandwidth exhausted by heavy data access workloads

Appendix

- our sysbench-xa repo (open source at https://github.com/zettadb/sysbench-xa) contains standard sysbench plus the following:
  - dedicated XA txn performance test cases(tests/db/*_xa.lua), as well as result curve generation script(get_sbout_stats.py)
  - all performance test raw results show in this slide and that of fomer runs in sysbench_xa/sysbench/results
  - All target db instance config files(PostgreSQL, Kunlun-storage and percona-server-8.0.22) can be found in sysbench-xa/sysbench/dbconfigs
- the kunlun-storage used in perf tests is enterprise edition(EE), community(open source) kunlun-storage may show performance not good as show here(althoug still much better than community mysql) because some performance enhancements in our EE are not open source, but binary version of EE can be downloaded in www.zettadb.com/downloads;
- kunlun-storage is developed based on percona-mysql-8.0.18; kunlun computing node software is developed based on postgresql-11.5.
- db instance configuration
  - kunlun-storage is developed based on percona-mysql-8.0.18; we compare performance with latest percona-server-8.0 version --- percona-server-8.0.22
  - we compared performance of kunlun-storage with percona-server-8.0.18 and percona-server-8.0.22 respectively before, results&analysis accessible in Chinese:
    - https://zhuanlan.zhihu.com/p/351042812 and https://zhuanlan.zhihu.com/p/151664455
  - for all db instances, flushes redo logs (and also binlogs for mysql variants) at each txn commit, and isolation level is READ COMMITTED
  - for all db instances, buffer pool size: 16GB; log buffer size: 256MB; all optimizer switches are default values
  - for each db instance, there is no replica, it runs standalone throughout the performance test
  - kunlun-storage and percona-mysql-8.0.22 both uses thread pools
  - for kunlun-storage and percona-mysql, binlog stored in separate SSD from innodb data dir containing innodb data files&WAL files;
  - for postgresql, WAL files&data files stored in same SSD; and vacuum&bgwriter&walwriter all use default settings.
- sysbench configuration
  - 20 tables each 4M rows, in all nearly 16GB data; mysql datadir test db dir took 18GB, postgresql datadir test db dir took 22GB;
  - important sysbench settings: num_threads=500; max-time= 600s; report-interval=1; max-requests=0; oltp-dist-type=uniform
  - 100 points sampled(1/6) in each test case's result to produce the result graphs, so the numbers shown in pictures for each curve is slightly different from results in the corresponding table
  - stddev of QPS and latency is computed using all 600 result entries for each case, so is QPS, avg/95% latency computed by sysbench itself.
  - oltp.lua case use default settings, i.e. 10 point selects and 4 other(range/sum/order/distinct) selects, and single row update/delete/insert located by PK: 1 index update, 1 non-index update, 1 insert and 1 delete; update_[non_]index[_xa].lua: one row update located by PK, indexed field or non-indexed field is updated;
  - sysbench run on same server as db instance to avoid network bottleneck
  - sysbench_xa xa test cases(oltp_xa, update_index_xa and update_non_index_xa): wrap same stmt with XA START/XA END;XA PREPARE;XA CO9MMIT; all use 2PC, and XA START/END/PREPARE/COMMIT commands not counted into QPS
- test machine configuration
  - One commodity server: AMD Ryzen7 3700x 16 cores; 48GB DDR 4 memory; 1TB + 1TB SSD; network card bandwidth: 1000Mb/s OS: Ubuntu-20.04 Linux 5.8.0-43-generic
  - network switch bandwidth: 1000Mb/s; network packet RTT: avg 0.1 ms, stddev: 0.6ms

Thank you!

Q&A