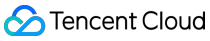
TDSQL for MySQL

Performance Test

Product Documentation

Copyright Notice

Trademark Notice



Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

Performance Test
TDStore Engine
TPC-C Test
Sysbench Test

Performance Test
TDStore Engine
TPC-C Test

Last updated: 2025-07-22 16:14:51

Test Overview

The TPC-C test report for the TDStore engine is provided as a performance comparison benchmark for different versions of the TDStore engine.

> **Note:**
> TPC-C is a common benchmark in the industry, developed and published by the TPC committee. It is used to evaluate the online transaction processing (OLTP-oriented) capabilities of databases. It primarily involves 10 tables and includes five different business transaction models: NewOrder (generation of new orders), Payment (order payment), OrderStatus (querying of recent orders), Delivery (delivery), and StockLevel (analysis of inventory stockout status). TPC-C uses the tpmC value (transactions per minute) to measure the system's max qualified throughput (MQTh). Transactions are determined by NewOrder, meaning that the final measurement unit is the number of new orders processed per minute.

Test Environment

Hardware Environment

Node Type	Node Specifications	Number of Nodes
Hybrid node	16-Core CPU/32 GB of Memory/Enhanced SSD 300 GB	3
Managing node	4Core CPU/8GB Memory	3

Software Version

Node Type	Software Version
Hybrid node	v20.0.0
BenchmarkSQL	v5.0

Parameter Configuration

```
set persist audit_log_policy = "NONE";
set persist max_prepared_stmt_count = 1000000;
set persist temptable_max_mmap = 214748364800
```

Testing Plan

Preparing Stress Test Tools

Download the open-source Benchmarksql stress test tool from the community and decompress it for testing.

Preparing Test Data

Perform a quantitative operation on 1,000 warehouses. Execute the following command on the stress test machine, modify the props.mysql configuration file, and fill in the corresponding instance connection information:

```
cd benchmarksql/run
vi props.mysql
```

The configuration file props.mysql and key parameters are as follows:

```
db=mysql
driver=com.mysql.jdbc.Driver
conn=jdbc:mysql://{ip}:{port}/tpcc?
useSSL=false&useServerPrepStmts=true&useConfigs=maxPerformance&rewriteBa
tchedStatements=true&cachePrepStmts=true&prepStmtCacheSize=1000&prepStmt
CacheSqlLimit=2048
user={user}
password={password}

warehouses=1000
loadWorkers=20

terminals={terminals}
runTxnsPerTerminal=0
runMins=5
limitTxnsPerMin=0
```

```
terminalWarehouseFixed=true

newOrderWeight=45
paymentWeight=43
orderStatusWeight=4
deliveryWeight=4
stockLevelWeight=4
```

Create a database:

```
 create database tpcc;
```

Start creating tables and indexes and modify the definition of the core table to a hash partitioned table. By default, 9 partitions are created.

```
cd benchmarksql/run/sql.common
# Modify the table structure. The core table is defined as a hash
partitioned table, which has 9 partitions by default.
cp tableCreates.sql  tableCreates_tdsql.sql
vi tableCreates_tdsql.sql

CREATE TABLE bmsql_config (
cfg_name    varchar(30) primary key,
cfg_value   varchar(50)
);


CREATE TABLE bmsql_warehouse (
   w_id        integer   not null,
   w_ytd       decimal(12,2),
   w_tax       decimal(4,4),
   w_name      varchar(10),
   w_street_1  varchar(20),
   w_street_2  varchar(20),
   w_city      varchar(20),
   w_state     char(2),
   w_zip       char(9),
   primary key(w_id)
) partition by hash(w_id) partitions 9;
```

```
CREATE TABLE bmsql_district (
    d_w_id       integer       not null,
    d_id         integer       not null,
    d_ytd        decimal(12,2),
    d_tax        decimal(4,4),
    d_next_o_id  integer,
    d_name       varchar(10),
    d_street_1   varchar(20),
    d_street_2   varchar(20),
    d_city       varchar(20),
    d_state      char(2),
    d_zip        char(9),
    PRIMARY KEY (d_w_id, d_id)
) partition by hash(d_w_id) partitions 9;

CREATE TABLE bmsql_customer (
    c_w_id         integer        not null,
    c_d_id         integer        not null,
    c_id           integer        not null,
    c_discount     decimal(4,4),
    c_credit       char(2),
    c_last         varchar(16),
    c_first        varchar(16),
    c_credit_lim   decimal(12,2),
    c_balance      decimal(12,2),
    c_ytd_payment  decimal(12,2),
    c_payment_cnt  integer,
    c_delivery_cnt integer,
    c_street_1     varchar(20),
    c_street_2     varchar(20),
    c_city         varchar(20),
    c_state        char(2),
    c_zip          char(9),
    c_phone        char(16),
    c_since        timestamp,
    c_middle       char(2),
    c_data         varchar(500),
    PRIMARY KEY (c_w_id, c_d_id, c_id)
) partition by hash(c_w_id) partitions 9;
```

```
CREATE TABLE bmsql_history (
    hist_id  integer,
    h_c_id   integer,
    h_c_d_id integer,
    h_c_w_id integer,
    h_d_id   integer,
    h_w_id   integer,
    h_date   timestamp,
    h_amount decimal(6,2),
    h_data   varchar(24)
) partition by hash(h_w_id) partitions 9;

CREATE TABLE bmsql_new_order (
    no_w_id  integer   not null ,
    no_d_id  integer   not null,
    no_o_id  integer   not null,
    PRIMARY KEY (no_w_id, no_d_id, no_o_id)
) partition by hash(no_w_id) partitions 9;

CREATE TABLE bmsql_oorder (
    o_w_id       integer      not null,
    o_d_id       integer      not null,
    o_id         integer      not null,
    o_c_id       integer,
    o_carrier_id integer,
    o_ol_cnt     integer,
    o_all_local  integer,
    o_entry_d    timestamp,
    PRIMARY KEY (o_w_id, o_d_id, o_id)
)  partition by hash(o_w_id) partitions 9;

CREATE TABLE bmsql_order_line (
    ol_w_id         integer   not null,
    ol_d_id         integer   not null,
    ol_o_id         integer   not null,
    ol_number       integer   not null,
    ol_i_id         integer   not null,
    ol_delivery_d   timestamp,
```

```
    ol_amount       decimal(6,2),
    ol_supply_w_id  integer,
    ol_quantity     integer,
    ol_dist_info    char(24),
    PRIMARY KEY (ol_w_id, ol_d_id, ol_o_id, ol_number)
) partition by hash(ol_w_id) partitions 9;

CREATE TABLE bmsql_item (
    i_id     integer      not null,
    i_name   varchar(24),
    i_price  decimal(5,2),
    i_data   varchar(50),
    i_im_id  integer,
    PRIMARY KEY (i_id)
);

CREATE TABLE bmsql_stock (
    s_w_id       integer       not null,
    s_i_id       integer       not null,
    s_quantity   integer,
    s_ytd        integer,
    s_order_cnt  integer,
    s_remote_cnt integer,
    s_data       varchar(50),
    s_dist_01    char(24),
    s_dist_02    char(24),
    s_dist_03    char(24),
    s_dist_04    char(24),
    s_dist_05    char(24),
    s_dist_06    char(24),
    s_dist_07    char(24),
    s_dist_08    char(24),
    s_dist_09    char(24),
    s_dist_10    char(24),
    PRIMARY KEY (s_w_id, s_i_id)
) partition by hash(s_w_id) partitions 9;
```

After the database and table are created, start importing stress test data:

```
cd ..
nohup ./runLoader.sh props.mysql &
```

After the data is imported, execute the following SQL to create an index:

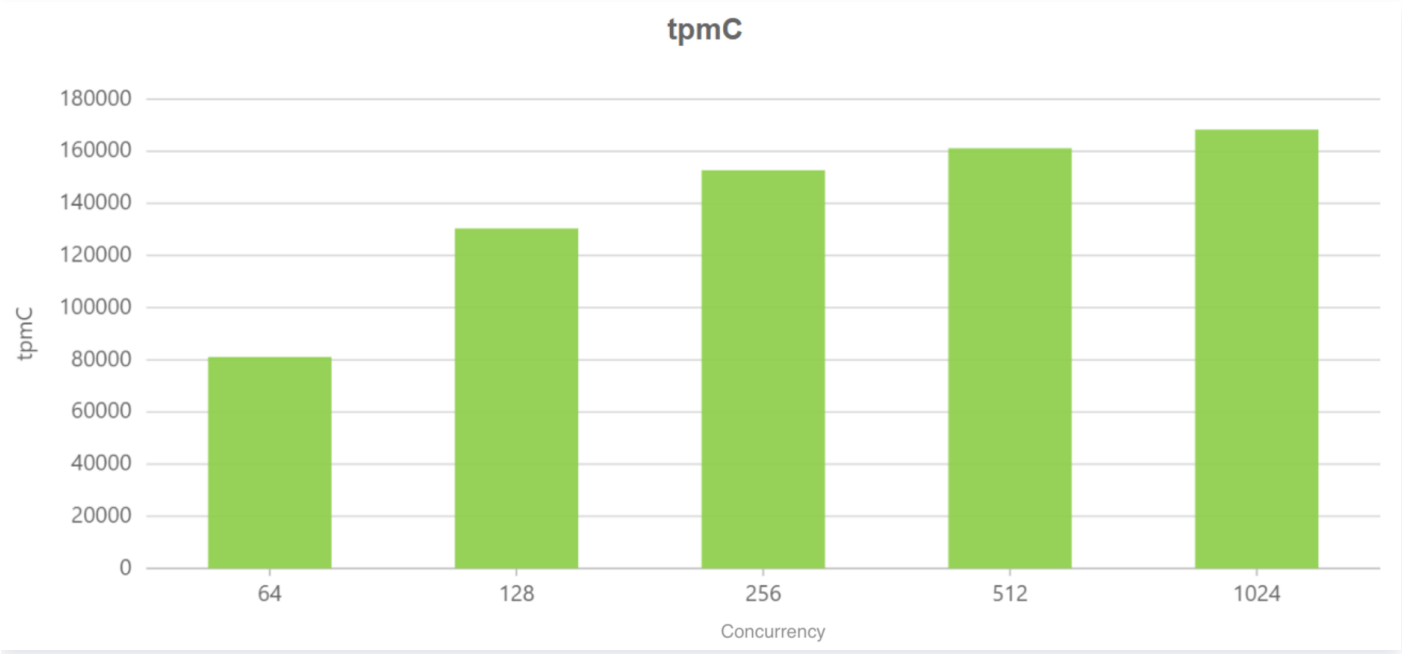
```
create index bmsql_customer_idx1 on bmsql_customer (c_w_id, c_d_id,
c_last, c_first);
create unique index bmsql_oorder_idx1 on bmsql_oorder (o_w_id, o_d_id,
o_carrier_id, o_id);
```

Executing a Test

Run the following command to run the TPC-C test:

```
cd benchmarksql/run
./runBenchmark.sh props.mysql
```

Test Result



Thread Count	tpmTOTAL	tpmC (NewOrders)
64	180266.52	81159.21
128	290460.47	130412.08
256	339837.92	152754.91

512	358118.36	161209.6
1024	374486.15	168361.04

Sysbench Test

Last updated：2025-07-22 16:15:09

Test Overview

The Sysbench test report of the TDStore engine is provided as a performance comparison benchmark for different versions.

Test Environment

Hardware Environment

Node Type	Node Specifications	Number of Nodes
Hybrid node	16-Core CPU/32 GB of Memory/Enhanced SSD 300 GB	3
Managing node	4Core CPU/8GB Memory	3

Software Version

Node Type	Software Version
HyperNode	v20.0.0
Sysbench	sysbench 1.1.0

Parameter configuration

```
set persist audit_log_policy = "NONE";
set persist max_prepared_stmt_count = 1000000;
set persist temptable_max_mmap = 214748364800
```

Testing Plan

Preparing Test Data

Initialize 32 tables, each with 10 million records.

```
sysbench oltp_common \
    --threads=256 \
    --mysql-host=xxxx \
```

```
    --mysql-port=xxxx \
    --mysql-user=root \
    --mysql-password=password \
    --mysql-db=sbtest \
    --auto_inc=off \
    --mysql-ignore-errors='all' \
    prepare --tables=32 --table-size=10000000
```

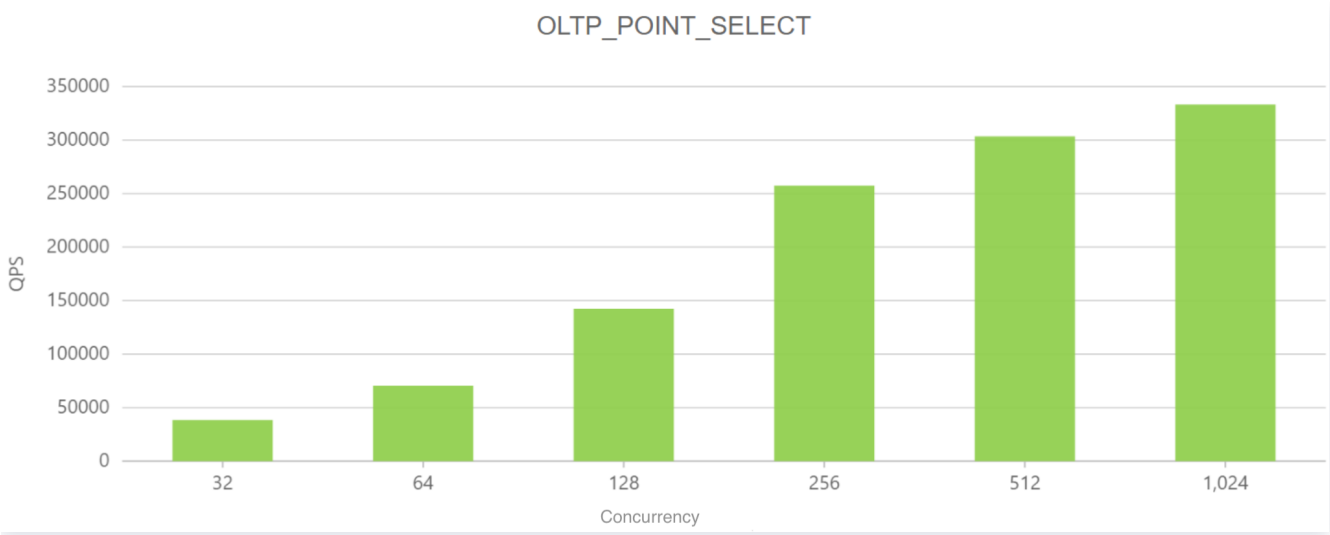
Executing a Test

Run the following commands to perform point queries, read-only, index updates, non-index updates, write-only, and mixed read-write OLTP scenario tests, respectively.

```
sysbench ${testname} \
    --threads=${threads} \
    --time=600 \
    --report-interval=1 \
    --mysql-host=xxxx \
    --mysql-port=xxxx \
    --mysql-user=root \
    --mysql-password=password \
    --mysql-db=sbtest \
    --auto_inc=off \
    --mysql-ignore-errors='all' \
    run --tables=32 --table-size=10000000
```

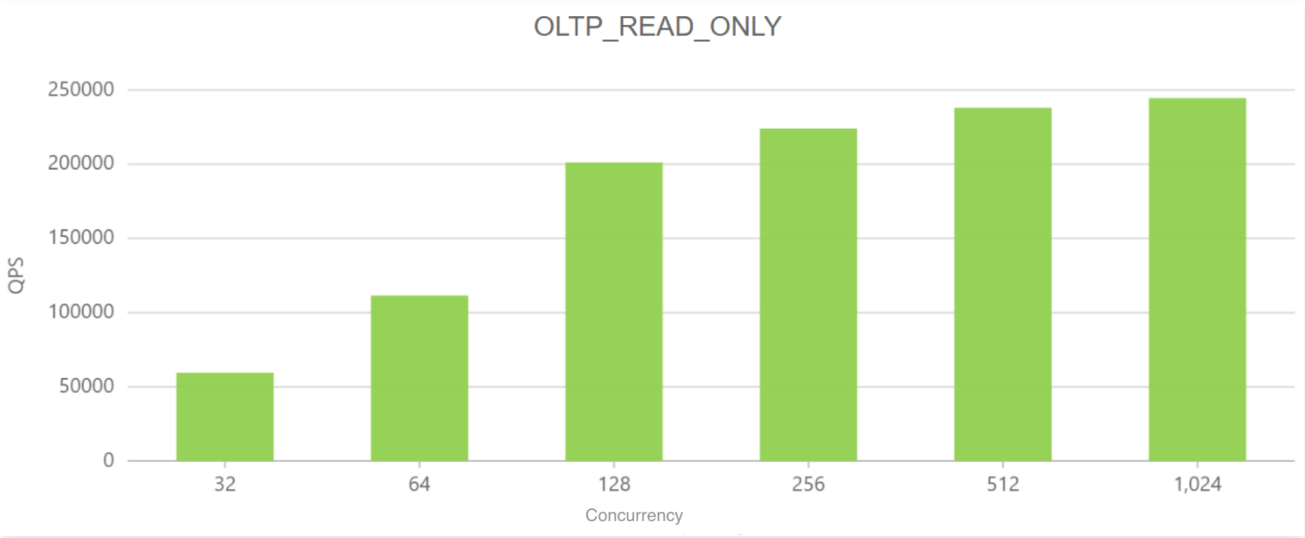
Test Result

Point Query Scenario



Thread Count	TPS	QPS	P95 Latency (ms)
32	38,529.83	38,529.83	1.08
64	70,432.26	70,432.26	1.16
128	142,401.56	142,401.56	1.14
256	257,455	257,455	1.3
512	303,421.66	303,421.66	1.96
1,024	333,089.14	333,089.14	12.08

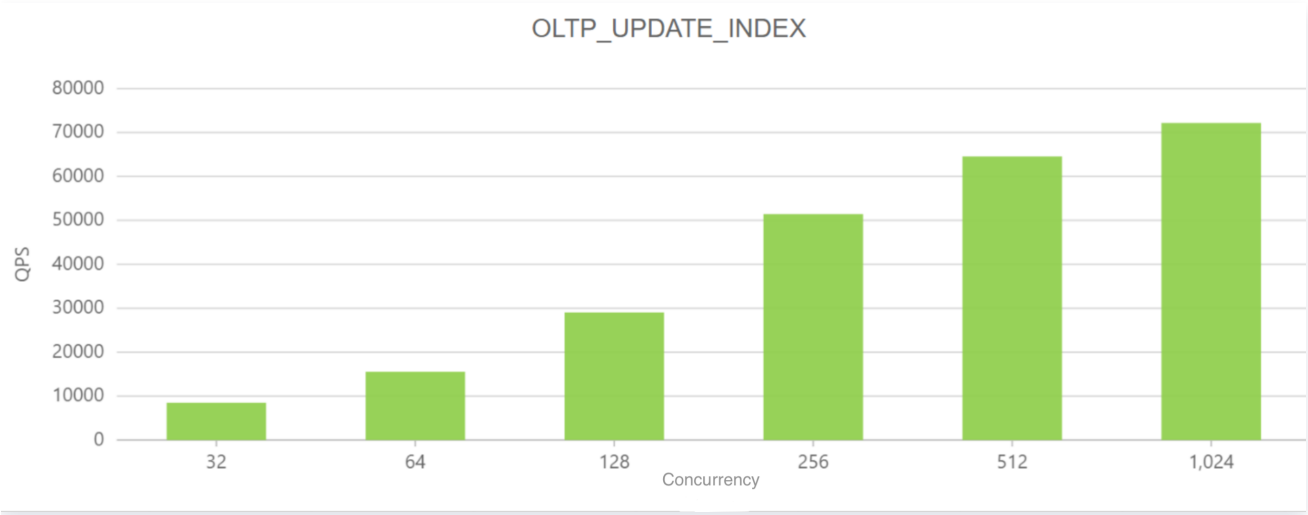
Read-Only Scenario



Thread Count	TPS	QPS	P95 Latency (ms)
32	3,718.16	59,490.55	10.27

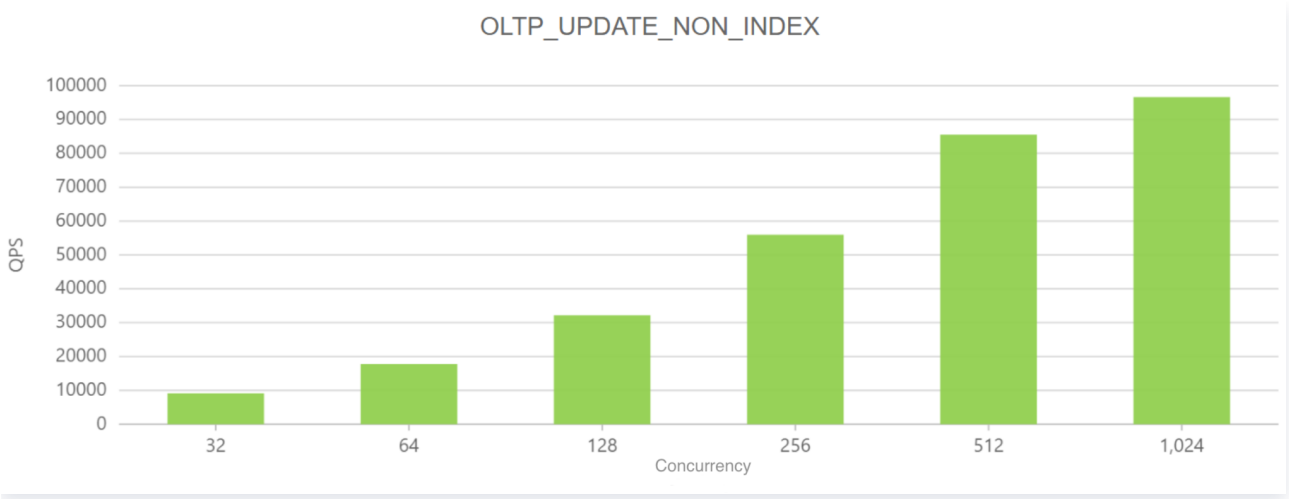
64	6,969.88	111,518	11.04
128	12,574.08	201,185.25	12.3
256	14,006.38	224,102.12	39.65
512	14,882.04	238,112.62	65.65
1,024	15,286.33	244,581.30	118.92

Index Update



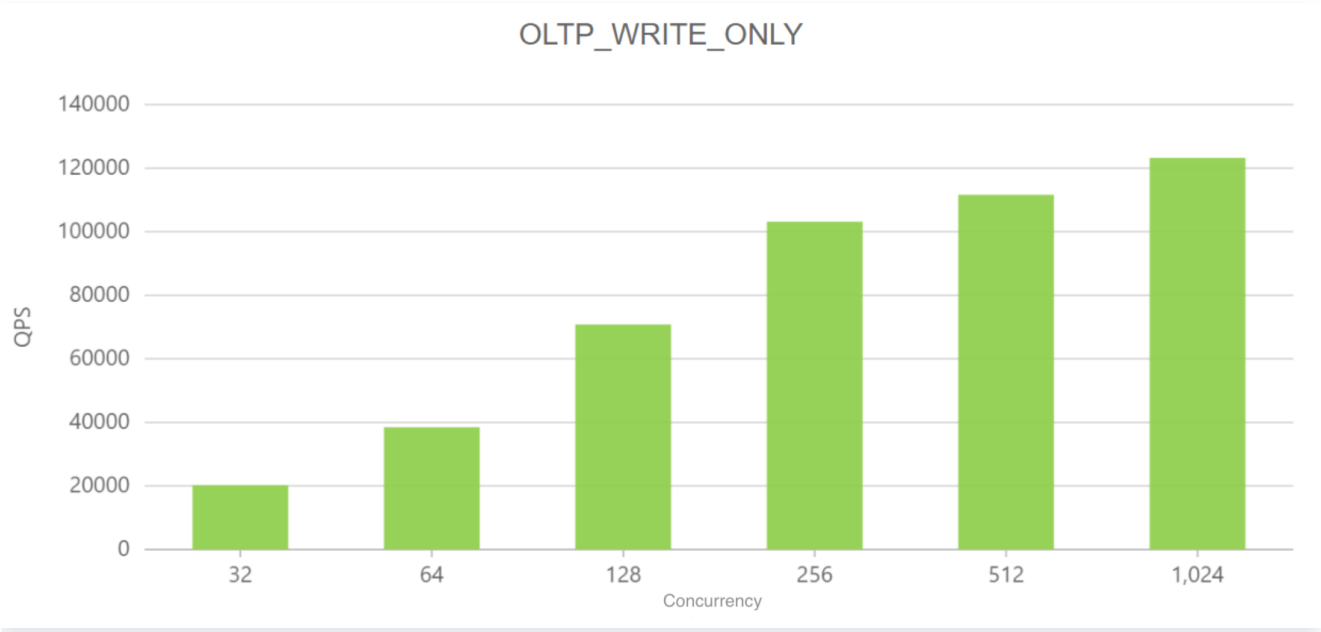
Thread Count	TPS	QPS	P95 Latency (ms)
32	8,505.21	8,505.21	5.67
64	15,562.77	15,562.77	6.09
128	29,050.14	29,050.14	6.55
256	51,415.89	51,415.89	7.56
512	64,561.69	64,561.69	19.65
1,024	72,160.95	72,160.95	44.98

Non-index Update



Thread Count	TPS	QPS	P95 Latency (ms)
32	9,088.07	9,088.07	5.67
64	17,736.05	17,736.05	5.57
128	32,157.99	32,157.99	6.21
256	55,961.77	55,961.77	7.04
512	85,495.51	85,495.51	11.24
1,024	96,605.66	96,605.66	25.74

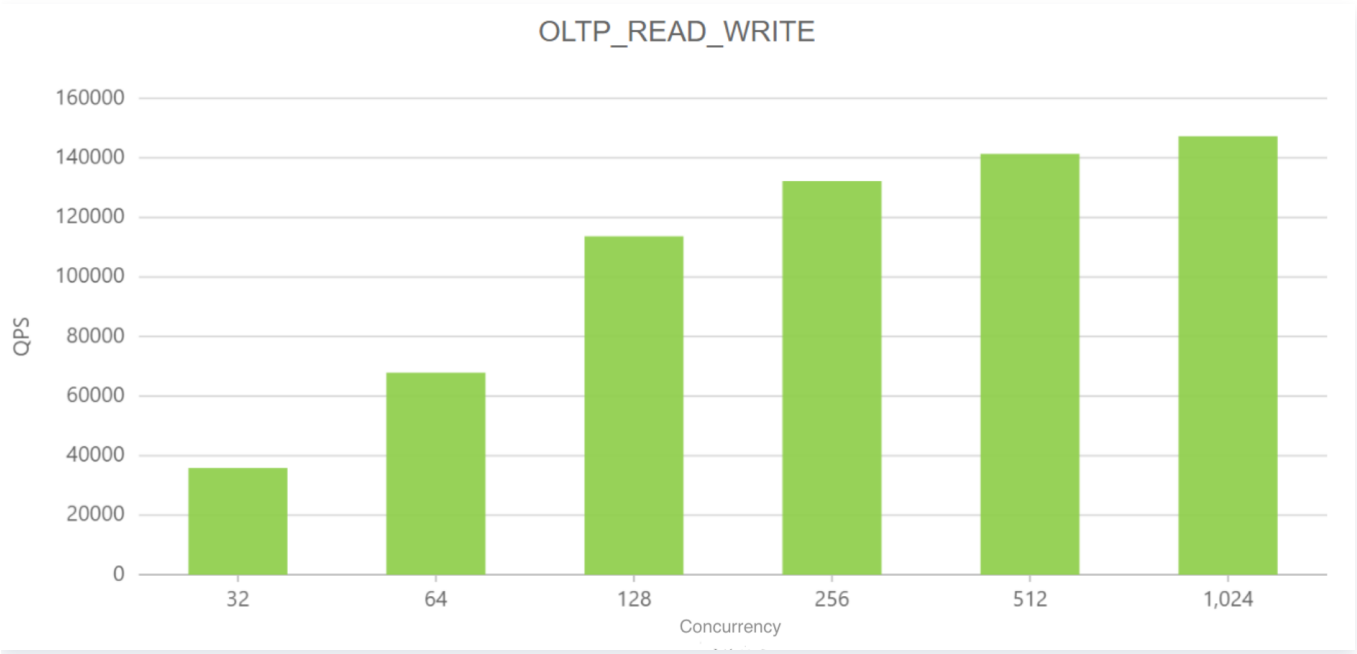
Write-Only Scenario



Thread Count	TPS	QPS	P95 Latency (ms)

32	3,366.87	20,201.25	13.46
64	6,410.02	38,460.15	13.95
128	11,800.35	70,802.10	15
256	17,188.05	103,128.33	25.28
512	18,600.43	111,602.61	56.84
1,024	20,536.86	123,221.14	89.16

Mixed Read-Write



Thread Count	TPS	QPS	P95 Latency (ms)
32	1,793.43	35,868.67	22.28
64	3,392.70	67,854.01	23.52
128	5,684.63	113,692.69	29.19
256	6,610.58	132,211.69	63.32
512	7,066.07	141,321.40	106.75
1,024	7,363.82	147,276.35	231.53