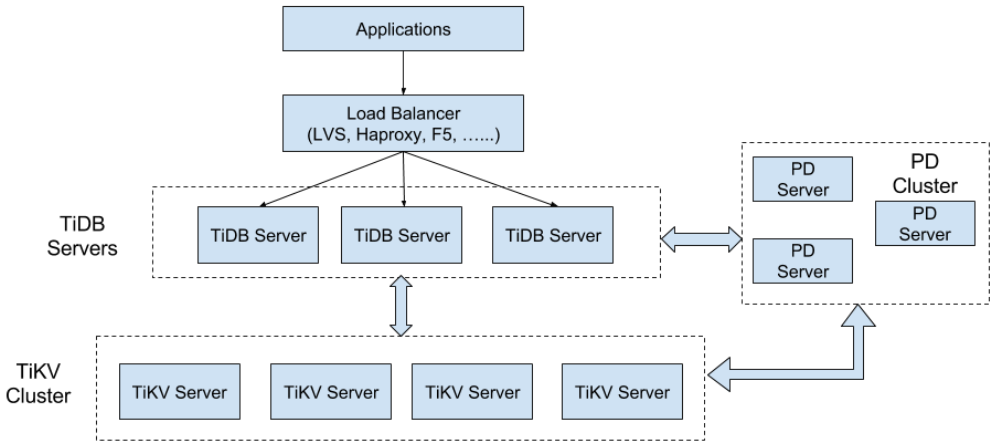
一、 TIDB 简介

TiDB 整体架构

要深入了解 TiDB 的水平扩展和高可用特点，首先需要了解 TiDB 的整体架构。



TiDB 集群主要分为三个组件：

TiDB Server

TiDB Server 负责接收 SQL 请求，处理 SQL 相关的逻辑，并通过 PD 找到存储计算所需数据的 TiKV 地址，与 TiKV 交互获取数据，最终返回结果。 TiDB Server 是无状态的，其本身并不存储数据，只负责计算，可以无限水平扩展，可以通过负载均衡组件（如 LVS、HAProxy 或 F5）对外提供统一的接入地址。

PD Server

Placement Driver (简称 PD) 是整个集群的管理模块，其主要工作有三个： 一是存储集群的元信息（某个 Key 存储在哪个 TiKV 节点）；二是对 TiKV 集群进行调度和负载均衡（如数据的迁移、Raft group leader 的迁移等）；三是分配全局唯一且递增的事务 ID。
PD 是一个集群，需要部署奇数个节点，一般线上推荐至少部署 3 个节点。

TiKV Server

TiKV Server 负责存储数据，从外部看 TiKV 是一个分布式的提供事务的 Key-Value 存储引擎。存储数据的基本单位是 Region，每个 Region 负责存储一个 Key Range （从 StartKey 到 EndKey 的左闭右开区间）的数据，每个 TiKV 节点会负责多个 Region 。TiKV 使用 Raft 协议做复制，保持数据的一致性和容灾。副本以 Region 为单位进行管理，不同节点上的多个 Region 构成一个 Raft Group，互为副本。数据在多个 TiKV 之间的负载均衡由 PD 调度，这里也是以 Region 为单位进行调度。

高度兼容 MySQL

大多数情况下，无需修改代码即可从 MySQL 轻松迁移至 TiDB，分库分表后的 MySQL 集群亦可通过 TiDB 工具进行实时迁移。

水平弹性扩展

通过简单地增加新节点即可实现 TiDB 的水平扩展，按需扩展吞吐或存储，轻松应对高并发、海量数据场景。

分布式事务

TiDB 100% 支持标准的 ACID 事务。

真正金融级高可用

相比于传统主从 (M-S) 复制方案，基于 Raft 的多数派选举协议可以提供金融级的 100% 数据强一致性保证，且在不丢失大多数副本的前提下，可以实现故障的自动恢复 (auto-failover)，无需人工介入。

说存储：https://pingcap.com/blog-cn/tidb-internal-1/
说计算：https://pingcap.com/blog-cn/tidb-internal-2/
谈调度：https://pingcap.com/blog-cn/tidb-internal-3/

二、 TIDB 与 MySQL 的差异

使用 mysql 客户端可以连接 tidb 集群，目前 TIDB 不支持以下特征：

存储过程
视图
触发器
自定义函数
外键约束
全文索引
空间索引
非 UTF8 字符集

与 MySQL 有差异的特性

自增 ID

TiDB 的自增 ID (Auto Increment ID) 只保证自增且唯一，并不保证连续分配。TiDB 目前采用批量分配的方式，所以如果在多台 TiDB 上同时插入数据，分配的自增 ID 会不连续。

注意：

在有多台 TiDB 使用自增 ID 时，建议不要混用缺省值和自定义值。因为目前在如下情况下

会报错：
在有两个 TiDB（TiDB A 缓存 [1,5000] 的自增 ID，TiDB B 缓存 [5001,10000] 的自增 ID）的集群，使用如下 SQL 语句创建一个带有自增 ID 的表：

```
create table t(id int unique key auto_increment, c int);
```

该语句执行如下：
客户端向 TiDB B 插入一条将 id 设置为 1 的语句，并执行成功。
客户端向 TiDB A 发送插入一条记录，且记录中 id 使用缺省值即 1，最终返回 Duplicated Error。

DDL 相关操作

以下操作不支持：
Add/Drop primary key 操作目前不支持。
Add Index/Column 操作不支持同时创建多个索引或列。
Drop Column 操作不支持删除的列为主键列或索引列。
Add Column 操作不支持同时将新添加的列设为主键或唯一索引，也不支持将此列设成 auto_increment 属性。
Change/Modify Column 操作目前支持部分语法，细节如下：
在修改类型方面，只支持整数类型之间修改，字符串类型之间修改和 Blob 类型之间的修改，且只能使原类型长度变长。此外，不能改变列的 unsigned/charset/collate 属性。这里的类型分类如下：
具体支持的整型类型有：TinyInt，SmallInt，MediumInt，Int，BigInt。
具体支持的字符串类型有：Char，Varchar，Text，TinyText，MediumText，LongText。
具体支持的 Blob 类型有：Blob，TinyBlob，MediumBlob，LongBlob。
在修改类型定义方面，支持的包括 default value，comment，null，not null 和 OnUpdate，但是不支持从 null 到 not null 的修改。
支持 LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE} 语法，但是不做任何事情（pass through）。
不支持对 enum 类型的列进行修改

```
mysql> alter table sbtest1 change test test_1 varchar(100) not null;
ERROR 1105 (HY000): unsupported modify column null to not null
```

```
mysql> alter table sbtest1 change test test_1 varchar(50) null;
ERROR 1105 (HY000): unsupported modify column length 50 is less than origin 100
```

事务

TiDB 使用乐观事务模型，在执行 Update、Insert、Delete 等语句时，只有在提交过程中才会检查写写冲突，而不是像 MySQL 一样使用行锁来避免写写冲突。所以业务端在执行 SQL 语句后，需要注意检查 commit 的返回值，即使执行时没有出错，commit 的时候也可能会出错。

Load data

语法：

```
LOAD DATA LOCAL INFILE 'file_name' INTO TABLE table_name
    {FIELDS | COLUMNS} TERMINATED BY 'string' ENCLOSED BY 'char' ESCAPED BY
'char'
    LINES STARTING BY 'string' TERMINATED BY 'string'
    (col_name ...);
```

其中 ESCAPED BY 目前只支持 '/\\'。

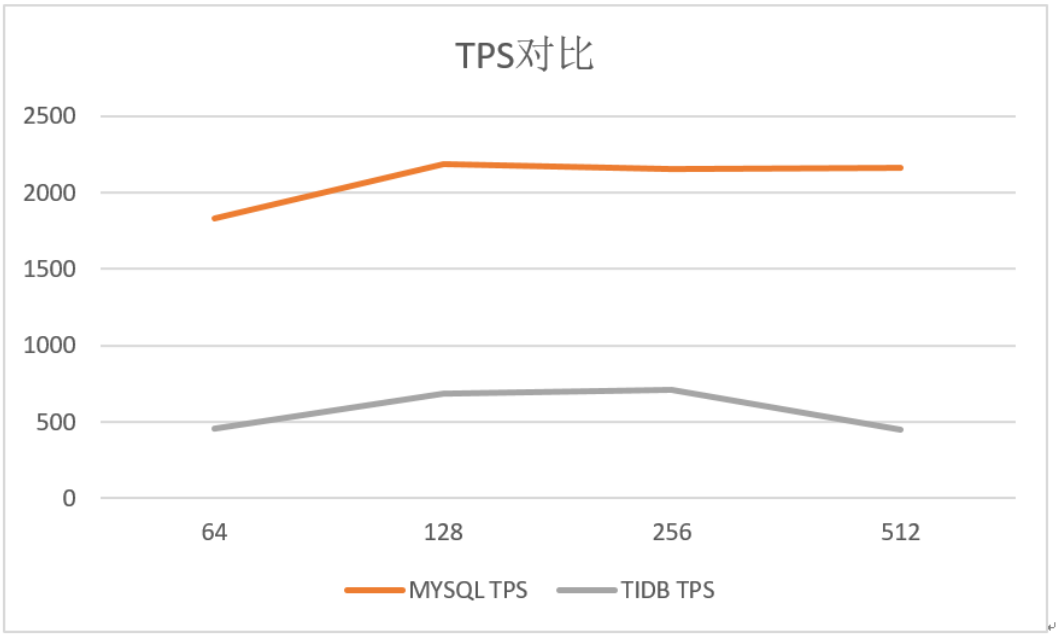
事务的处理：

TiDB 在执行 load data 时，默认每 2 万行记录作为一个事务进行持久化存储。如果一次 load data 操作插入的数据超过 2 万行，那么会分为多个事务进行提交。如果某个事务出错，这个事务会提交失败，但它前面的事务仍然会提交成功，在这种情况下一次 load data 操作会有部分数据插入成功，部分数据插入失败。而 MySQL 中会将一次 load data 操作视为一个事务，如果其中发生失败情况，将会导致整个 load data 操作失败。

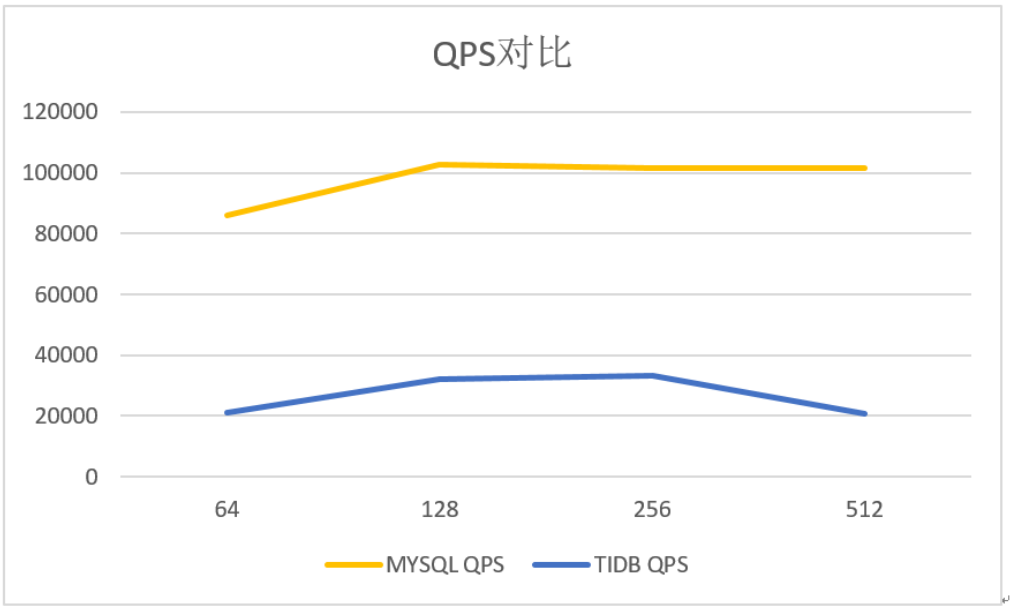
测试机器硬件配置如下：

组件	CPU	内存	本地存储	网络	实例数量(最低要求)
TiDB	16 核	24 GB	SAS, 500 GB	千兆网卡	2（一台独立部署，另外一台和监控混布）
PD	8 核	24 GB	SSD, 300 GB	千兆网卡	3
TiKV	16 核	32 GB	SSD, 600 GB	千兆网卡	4(三台独立部署，一台安装 mysql 用作对比测试)
监控	8 核	24 GB	SAS, 500 GB	千兆网卡	1（可以和 TiDB 混布）
				服务器总计	9（混布情况下最少机器数）

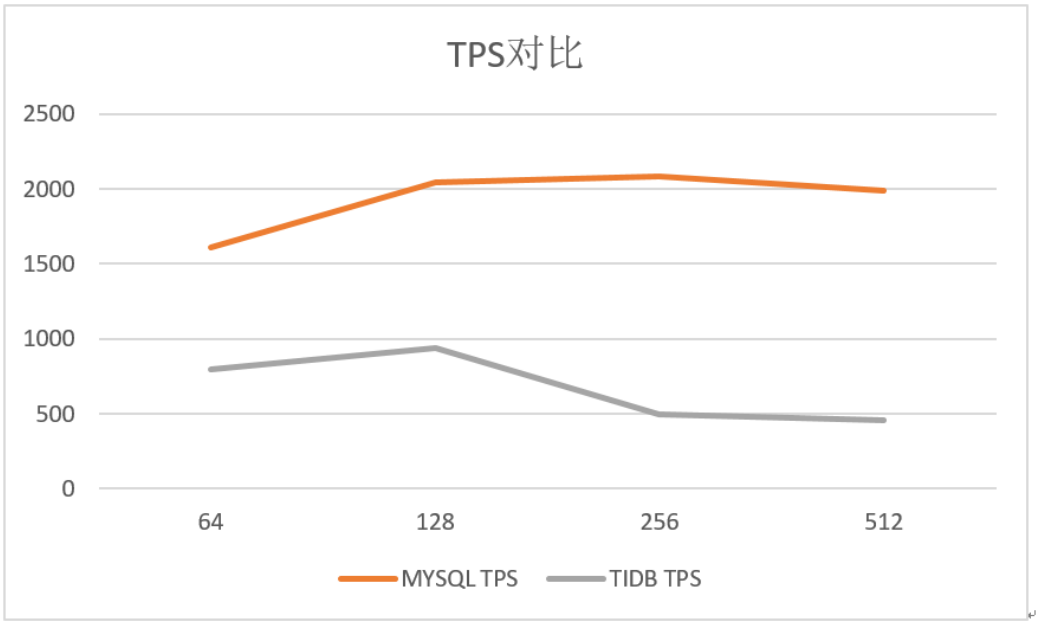
TPS 对比曲线图如下：


TPS对比
2500
2000
1500
1000
500
0
64
128
256
512
MYSQL TPS
TIDB TPS

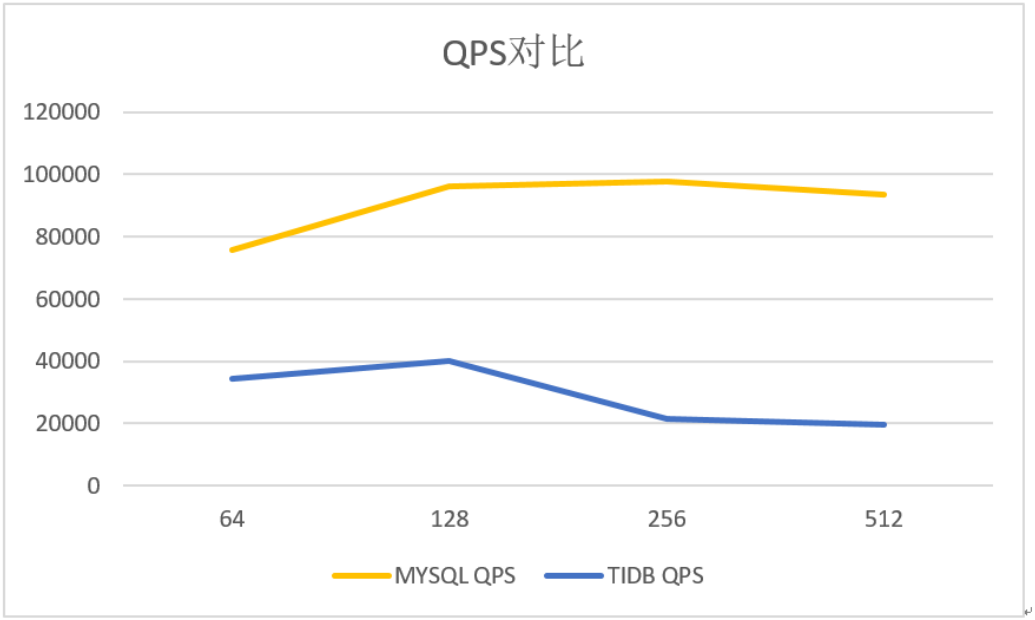
QPS 对比曲线图如下：


QPS对比
120000
100000
80000
60000
40000
20000
0
64
128
256
512
MYSQL QPS
TIDB QPS

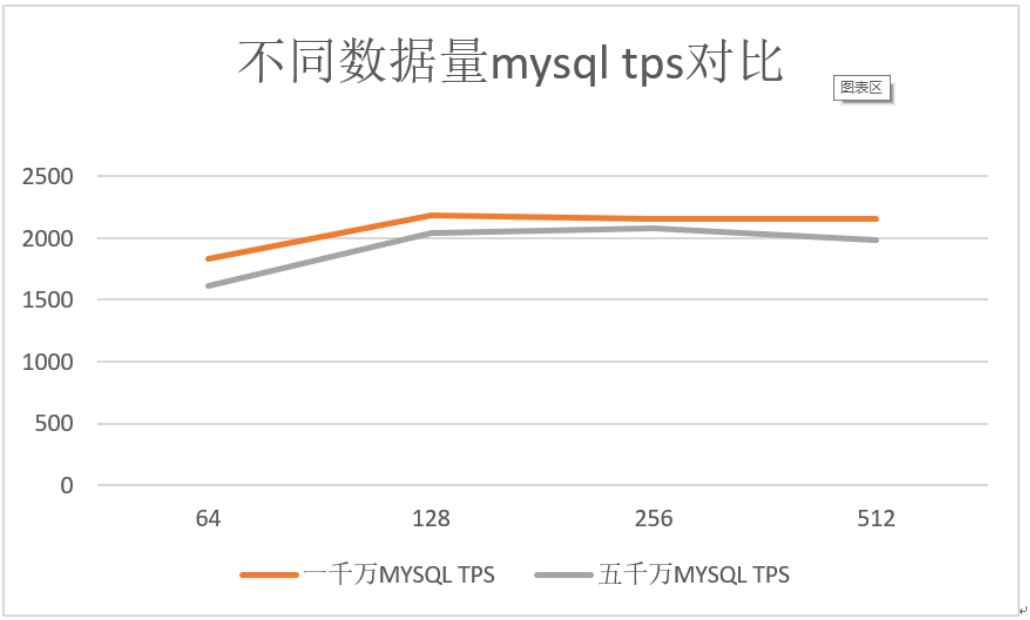
TPS 对比曲线图如下：

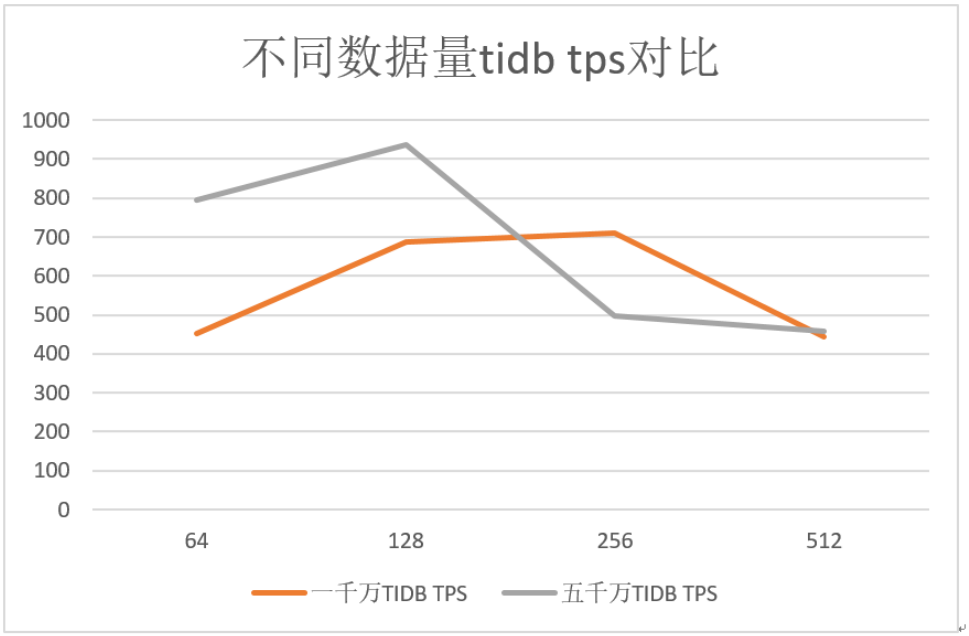


QPS 对比曲线图如下：

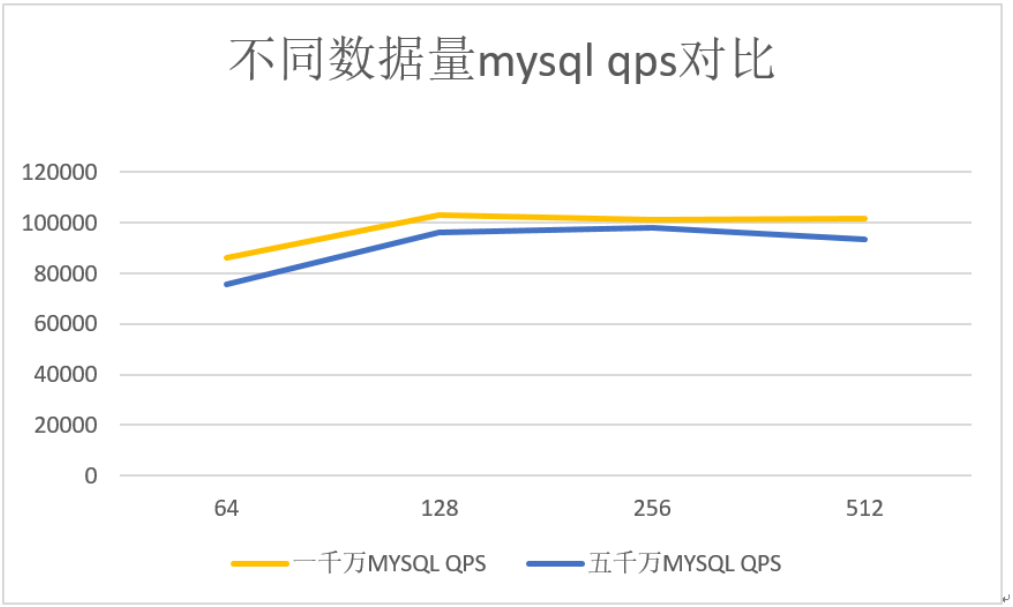


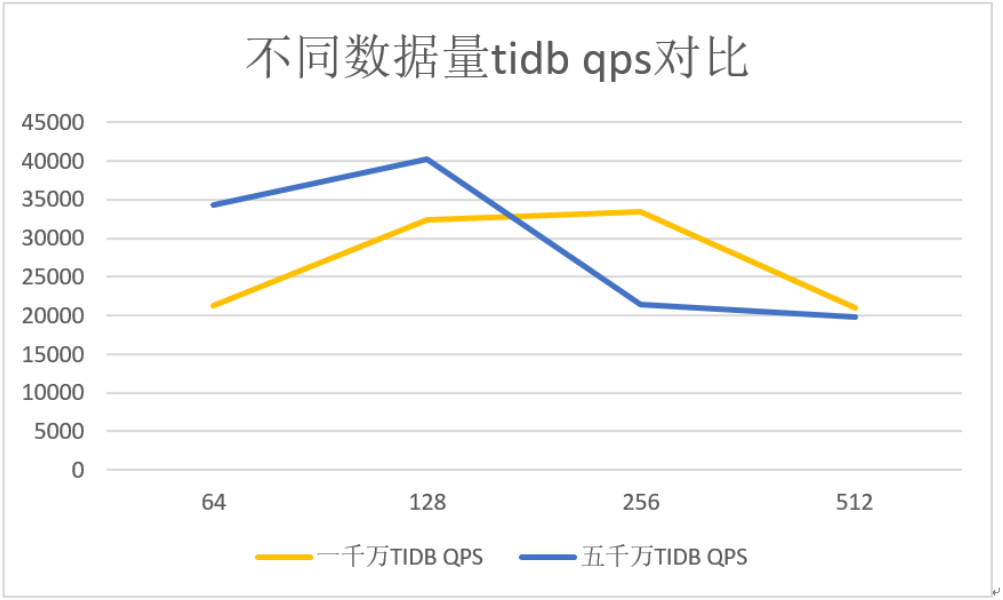
同类数据库在不同数据量的TPS对比





同类数据库在不同数据量的 QPS 对比





2、DDL 测试

drop index k_1 on sbtest1;　在一千三百万数据的时候执行了1.02 sec
create index k_1 on sbtest1(k);出现客户端异常 ERROR 2013 (HY000): Lost connection to MySQL server during query

```
mysql> create index k_1 on sbtest1(k);
ERROR 2013 (HY000): Lost connection to MySQL server during query
```

使用show processlist;查看create index k_1 on sbtest1(k);这个脚本还在数据库中执行

```
mysql> show processlist;
+------+----------+------------+---------+---------+------+-------+--------------------------------+------+
| Id   | User     | Host       | db      | Command | Time | State | Info                           | Mem  |
+------+----------+------------+---------+---------+------+-------+--------------------------------+------+
| 1224 | hjdbahdd | 10.51.0.89 | test330 | Query   |    0 | 2     |                                |    0 |
| 1225 | root     | 10.51.0.89 | test330 | Query   |  804 | 2     | create index k_1 on sbtest1(k) |    0 |
| 1226 | root     | 10.51.0.89 | test330 | Query   |    0 | 2     | show processlist               |    0 |
+------+----------+------------+---------+---------+------+-------+--------------------------------+------+
3 rows in set (0.00 sec)
```

经过排查这个错误是由于 haproxy 配置了过短的超时时间导致，调整超时时间以后不在出现。

三千万数据共耗时 1 hour 3 min 49.70 sec

alter table sbtest1 add test varchar(20) not null default 'aa'; 在两千一百万并且持续写入的情况下执行 0.60 sec

alter table sbtest1 change column test test_1 varchar(50) not null default 'aa'; 在五千万数据的情况下执行 0.11 sec

alter table sbtest1 modify column pad char(100); 在五千万数据的情况下执行 0.11 sec

alter table sbtest1 drop test_1 ; 在五千万数据的情况下执行 0.33 sec

alter table sbtest1 rename to sbtest10; 在五千万数据的情况下执行 0.11 sec

truncate table sbtest10; 在五千万数据的情况下执行 0.16 s