TDSQL MySQL版

实践教程

腾讯云

文档目录

实践教程
从单机实例导入到分布式实例

最近更新时间：2024-04-25 16:08:01

由于分布式数据库的分布式架构对用户透明，一般情况下，只需要预先建好表结构。可以使用 mysqldump、或其他 Navicat、SQLyog 等 MySQL 客户端进行迁移。迁移步骤如下：
第一步：准备导入导出环境
第二步：导出源表的表结构和数据
第三步：修改建表语句并在目的表中创建表结构
第四步：导入数据

步骤1：准备导入导出环境

准备迁移数据前，您需要准备好如下环境：

- 云服务器
  - 建议配置 CPU2核，内存8GB，磁盘500GB以上（取决于数据量大小）
  - Linux
  - 安装 MySQL 客户端
  - 如果您迁移的数据量较小（<10GB），也可以通过外网（互联网）直接导入，无需准备。
- TDSQL MySQL 版
  - 根据预期选择大小，并根据源库字符集，表名大小写，innodb_page_size 大小进行初始化。
  - 创建账号，该账号建议开启全局所有权限。
  - 必要时开启外网 IP

步骤2：导出源表的表结构和数据

演示环境

- 操作库：caccts
- 操作表：t_acct_water_0
- 源库：单实例 MySQL
- 目标库：TDSQL for Percona、MariaDB。

在源库导出表结构

通过命令

```
mysqldump -u username -p password -d dbname tablename -S socketpath > tablename.sql
```

导出表结构。

```
//命令实例
mysqldump -utest -ptest1234 -d -S /data/4003/prod/mysql.sock caccts
t_acct_water_0 > table.sql
```

在源库导出表数据

通过命令

`mysqldump -c -u username -p password dbname tablename -S socketpath > tablename.sql`

导出表数据。

```
//命令实例
mysqldump -c -t -utest -ptest1234  -S /data/4003/prod/mysql.sock caccts
t_acct_water_0 > data.sql
```

> ⚠ **注意：**
> 导出数据必须通过 mysqldump 工具导出，并且加上 -c 参数，因为这样导出的数据行都带有列名字段，不带列名字段的 sql 会被 TDSQL for Percona、MariaDB 拒绝掉。-t 参数的意义是不导出表结构，只导出表数据。

上传文件至云服务器某目录

上传前，您需开启 CVM 外网访问地址，并参见 Linux 系统通过 SCP 上传文件到 Linux 云服务器 上传文件，您至少需要上传刚刚导出的：

- 表结构 sql：table.sql
- 数据 sql：data.sql

步骤3：修改建表语句并在目的表中创建表结构

打开刚导出的表结构文件 table.sql，参考如下格式语句添加主键和 shardkey，并另存为 tablenew.sql。

```
CREATE TABLE (
列名称1 数据类型,
列名称2 数据类型,
列名称3 数据类型,
....,
PRIMARY KEY('列名称n'))
ENGINE=INNODB DEFAULT CHARSET=xxxx
shardkey=keyname
```

> ⚠ **注意：**

必须要设置主键，必须指定 shardkey，必须注意表名大小写问题，建议删除多余注释，否则建表可能不成功。

```
CREATE TABLE `t_acct_water_0` (
  `Fseq_no` int(11) NOT NULL DEFAULT '0',
  `Facct_id` varchar(64) NOT NULL DEFAULT '',
  `Facct_key` varchar(128) NOT NULL DEFAULT '',
  `Ffirstkey` varchar(64) NOT NULL DEFAULT '',
  `Fuin` varchar(64) NOT NULL DEFAULT '',
  `Fuserid` varchar(64) NOT NULL DEFAULT '',
  `Factionid` varchar(32) NOT NULL DEFAULT '',
  `Fzoneid` varchar(32) NOT NULL DEFAULT '',
  `Froleid` varchar(32) NOT NULL DEFAULT '',
  `Fsubacctid` varchar(64) NOT NULL DEFAULT '',
  `Fextern_tran_type` int(11) NOT NULL DEFAULT '0',
  `Fbase_tran_type` int(11) NOT NULL DEFAULT '0',
  `Fwater_type` int(11) NOT NULL DEFAULT '0',
  `Fio_flag` int(11) NOT NULL DEFAULT '0',
  `Fbill_no` varchar(64) NOT NULL DEFAULT '',
  `Fportal_seq` varchar(256) NOT NULL DEFAULT '',
  `Ftran_amt` bigint(20) NOT NULL DEFAULT '0',
  `Fbalance` bigint(20) NOT NULL DEFAULT '0',
  `Fgen_tran_amt` bigint(20) NOT NULL DEFAULT '0',
  `Fgen_balance` bigint(20) NOT NULL DEFAULT '0',
  `Fcreate_time` int(11) NOT NULL DEFAULT '0',
  `Fsource` varchar(32) NOT NULL DEFAULT '',
  `Fdevice` varchar(32) NOT NULL DEFAULT '',
  `Fcheck_account` varchar(256) NOT NULL DEFAULT '',
  `Fclient_ip` varchar(32) NOT NULL DEFAULT '',
  `Fuser_ip` varchar(32) NOT NULL DEFAULT '',
  `Ftran_info` varchar(256) NOT NULL DEFAULT '',
  `Fremark` varchar(256) NOT NULL DEFAULT '',
  `Freserve1` varchar(256) NOT NULL DEFAULT '',
  `Freserve2` varchar(256) NOT NULL DEFAULT
  `blobtest` blob,
  PRIMARY KEY (`Fseq_no`),
  KEY `Ffirstkey` (`Ffirstkey`),
  KEY `s_acctid_acctkey` (`Facct_id`,`Facct_key`),
  KEY `i_create_time` (`Fcreate_time`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 shardkey=Fseq_no
```

步骤4：导入数据

连接 TDSQL for Percona、MariaDB 实例

在 CVM 上使用 `mysql -u username -p password -h IP -P port` 登录 MySQL 服务器，然后使用 `use dbname` 进入数据库。

⚠ 注意：
您可能需要先创建库。

导入表结构

使用刚刚上传的文件，用 `source` 命令导入数据。

1. 先导入表结构： `source /文件路径/tablenew.sql;`

2. 再导入数据：`source /文件路径/data.sql;`
3. 校验导入情况：`select  count(*) from tablename;`

> **注意：**
> 需先导入建表语句，再导入数据。也可以通过 mysql 的 source 命令直接导入 sql。

其他方案

整体来说，只要能够在导入数据前，在目的表先创建对应的表结构（需指定 shardkey），就可以比较顺利的导入数据。

从分布式实例导入到分布式实例

最近更新时间：2023-12-22 11:27:11

由于分布式数据库到分布式数据库的数据导入方案与一般情况不同，使用 mysqldump 对分布式实例导入数据到分布式实例的步骤如下：

1. 安装 MariaDB 版本的 mysqldump

购买 Linux 版的云服务器，使用 `yum install mariadb-server` 即可安装。

2. 导出表结构

```
mysqldump --compact --single-transaction -d -uxxx -pxxx -
hxxx.xxx.xxx.xxx -Pxxxx db_name table_name > schema.sql
```

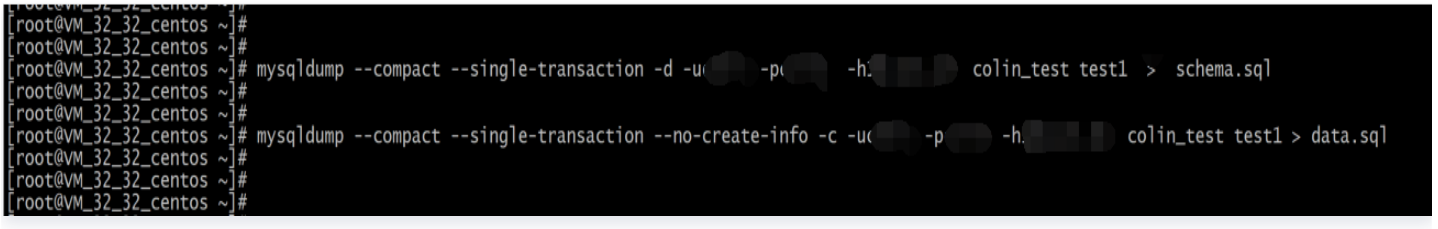
> **说明：**
> db_name 和 table_name 参数根据实际情况选择（db_name是库，table_name是表）。

3. 导出数据

mysqldump 导出数据：
在 TDSQL MySQL版 控制台 的参数设置中设置 net_write_timeout 参数：set global net_write_timeout=28800。

```
mysqldump --compact --single-transaction --no-create-info -c -uxxx -
pxxx -hxxx.xxx.xxx.xxx -Pxxxx db_name table_name > data.sql
```

> **说明：**
> db_name 和 table_name 参数根据实际情况选择，如果导出的数据要导 到另外 套 TDSQL MySQL版 环境的话，必须加上 -c 选项，-c 与 db_name 之间需添加空格。



4. 在目标库创建 db

```
mysql --default-character-set=utf8 -uxxx -pxxx -hxxx.xxx.xxx.xxx -Pxxxx
-e "create database dbname;";
```

- --default-character-set=utf8：根据您目标表实际情况设定。
- -uxxx：有权限的账号（-u 是关键字）。
- -pxxx：密码（-p 是关键字）。
- -hxxx.xxx.xxx.xxx -Pxxxx：数据库实例的 IP 和端口。
- dbname：代表 db 的名字。

5. 在目标库上导入表结构

```
mysql --default-character-set=utf8 -uxxx -pxxx -hxxx.xxx.xxx.xxx -Pxxxx
dbname < schema.sql
```

- --default-character-set=utf8：根据您目标表实际情况设定。
- -uxxx：有权限的账号（-u 是关键字）。
- -pxxx：密码（-p 是关键字）。
- -hxxx.xxx.xxx.xxx -Pxxxx：数据库实例的 IP 和端口。
- dbname：代表 db 的名字。

6. 在目标库上导入表数据

```
mysql --default-character-set=utf8 -uxxx -pxxx -hxxx.xxx.xxx.xxx -Pxxxx
dbname < data.sql
```

说明：
如果源表中使用了自增字段，并且导入的时候出现“Column 'xx' specified twice”的错误，则需要对 schema.sql 做处理。去掉自增字段的反引号(cat schema.sql | tr "\`" " " > schema_tr.sql)，然后 drop database，使用处理后的 schema_tr.sql 重复步骤3 – 5的操作。

```
[root@VM_32_32_centos ~]#
[root@VM_32_32_centos ~]# mysql --default-character-set=utf8 -u[redacted] -p[redacted] -h[redacted] -e "create database colin_test;"
[root@VM_32_32_centos ~]#
[root@VM_32_32_centos ~]#
[root@VM_32_32_centos ~]# mysql --default-character-set=utf8 -u[redacted] -p[redacted] -h[redacted] colin_test <  schema.sql
[root@VM_32_32_centos ~]#
[root@VM_32_32_centos ~]#
[root@VM_32_32_centos ~]#
[root@VM_32_32_centos ~]# mysql --default-character-set=utf8 -u[redacted] -p[redacted] -h[redacted] colin_test <  data.sql
[root@VM_32_32_centos ~]#
```

选择实例配置和分片配置

最近更新时间：2024-07-03 11:02:01

TDSQL MySQL 版选型概述

TDSQL MySQL 版由分片（sharding）组成，分片的规格和分片数量决定了 TDSQL MySQL 版实例的处理能力。理论上来讲：

- TDSQL MySQL 版实例读写并发性能 = ∑（某规格分片性能 * 某规格分片数量）
- TDSQL MySQL 版实例事务性能 = ∑（某规格分片事务性能 * 70% * 某规格分片数量）

因此，分片规格越高、分片数量越多，实例的处理能力越强。而分片性能，主要与 CPU / 内存 相关，并以 QPS 为基础衡量指标，我们在分片性能说明章节，给出了大致性能指标。

TDSQL MySQL 版分片规格的选择

TDSQL MySQL 版分片规格的选择，主要从三个方面需求来决定：1、性能需求；2、容量需求；3、其他要求。
性能需求：通过预判至少6个月的性能规模和可能增长，您可以确定您分布式实例所需总 CPU / 内存 规模。
容量需求：通过预判至少1年的容量规模和可能增长，您可以确定您分布式实例所需总 磁盘 规模。
其他要求：我们建议一个分片**至少存储5000万行数据**，并考虑到业务中所需的 广播表、单表，和节点内 join 等业务需求。

> ⚠ **注意：**
> 建议您先确保让单个分片配置较大，而分片数量较少。

综合上述来看，我们预估您可能有如下几种业务特点，推荐策略如下：

- 使用 TDSQL MySQL 版做功能性测试，且对性能没有特别要求：2个分片，每个分片配置为：**内存/磁盘：2GB/25GB**。
- 业务发展初期，总数据规模较小但增长快的选型：2个分片，每个分片配置为：**内存/磁盘：16GB/200GB**。
- 业务发展稳定，根据业务实际情况选型：4个分片，每个分片配置等于：**当前业务峰值*增长率/4**。

TDSQL MySQL 版分片性能测试

数据库基准性能测试为 sysbench 0.5 工具。此性能测试为针对单个分片的测试。
修改说明：对 sysbench 自带的 oltp 脚本做了修改，读写比例修改为 4：1，并通过执行测试命令参数 oltp_point_selects 和 oltp_index_updates 控制读写比例，本文测试用例的均采用4个 select 点，1个 update 点，读写比例保持 4:1。

内存 (GB)	存储空间 (GB)	数据集 (GB)	客户端数	单客户端并发数	QPS	TPS
2GB	100GB	46GB	1	128	1880	351

4GB	200GB	76GB	1	128	3983	797
8GB	200GB	142GB	1	128	6151	1210
16GB	400GB	238GB	1	128	10098	2119
32GB	700GB	238GB	2	128	20125	3549
64GB	1T	378GB	2	128	37956	7002
96GB	1.5T	378GB	3	128	51026	10591
120GB	2T	378GB	3	128	81050	15013
240GB	3T	567GB	4	128	96891	17698
480GB	6T	567GB	6	128	140256	26599

此处 TPS 为单机 TPS，并非测试的是分布式事务的 TPS。
根据运营策略要求，当前 TDSQL MySQL 版的（部分）实例都采用闲时超用技术，所以您可能在您的监控中看到 CPU 利用率超过100%的情况。

Online DDL 的技术演进与使用实践

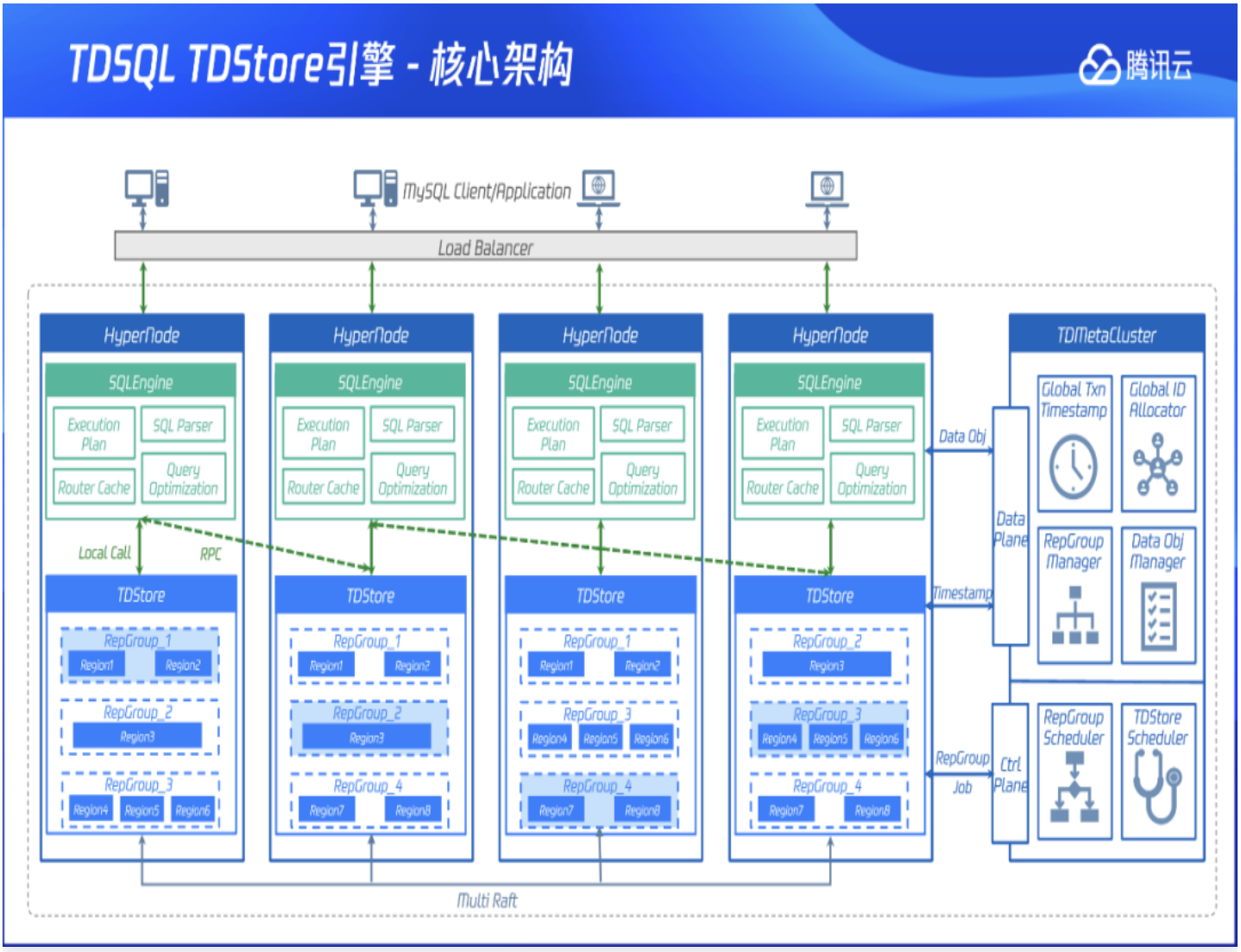
最近更新时间：2025-04-22 15:40:02

引言

在数据库的发展历史中，DDL (Data Definition Language) 操作一直是一个重要的挑战。在主流商业数据库的早期版本中，传统的 DDL 操作，往往都需要对数据表进行锁定，以防止在操作过程中发生数据不一致。这种锁定操作会导致数据库在 DDL 操作期间无法提供服务，对于需要7 * 24运行的大型现代应用来说，这是难以接受的。
为了解决这个问题，ORACLE 自 9i 引入 Online Table Redefinition，MySQL 自5.6引入 Online DDL，都旨在显著减少（或消除）更改数据库对象所需的应用程序停机时间（该特性以下统称 Online DDL）。随后，MySQL 8.0进一步引入了 Instant 算法，这是 DDL 操作的一个重大突破。Instant DDL 允许立即执行某些 DDL 操作，如尾部添加字段、修改字段名、修改表名等等，均无需锁定表或复制数据。这大大提高了 DDL 操作的速度，并减少了对业务的影响。尽管如此，在某些情况下 DBA 可能仍然需要依赖像 pt-online-schema-change(pt-osc) 这样的外部工具来执行 DDL 操作。这是因为 Online DDL 并不支持所有类型的 DDL 操作，同时 pt-osc 的流控等功能可以提供更多的灵活性。
随着分布式数据库的兴起，Online DDL 面临了新的挑战。在分布式环境中，DDL 操作需要在多个节点上同时进行，不仅需要保证 DDL & DML 操作并发的安全性，同时还要考虑性能、执行效率以及 crash-safe 问题。
本文将分享腾讯云数据库 TDSQL 系列的最新产品：TDStore 的 Online DDL 的技术演进与使用实践。

TDStore 架构介绍

TDStore 是腾讯面向金融级应用场景的，高性能、高可用的企业级分布式数据库解决方案，采用容器化云原生架构，提供集群高性能计算能力和低成本海量存储。
下图是 TDStore 的架构图，可以看到整个实例分为两类节点：



- **右侧的 TDMetaCluster 是管控节点**：负责集群的元数据管理、智能化调度。
- **左侧的 HyperNode 是对等节点，或者叫做混合节点**：每个节点分为计算层 SQLEngine 和存储层 TDStore 两个部分，其中 SQLEngine 层负责 SQL 解析、查询计划生成、查询优化等，它向下层的 TDStore 发送事务相关的读写请求。

TDStore 架构和功能特性：全分布式 + 存算一体/存算分离 + 数据面/控制面分离 + 高可扩展 + 全局一致性 + 高压缩率。

TDStore Online DDL 的难题攻克

当下传统单机 MySQL 执行 DDL 的思路：

1. instant DDL（ALGORITHM=INSTANT）：只需修改数据字典中的元数据，无需拷贝数据也无需重建表，原表数据不受影响。对此类 DDL 语句，可直接执行，瞬间完成。
2. inplace DDL（ALGORITHM=INPLACE）：存储引擎层"就地"重建表，虽然不阻塞 DML 操作，但对于大表变更可能导致长时间的主从不一致。对此类 DDL 语句，如果想使 DDL 过程更加可控，且对从库延迟比较敏感，建议使用第三方在线改表工具 ptosc 完成。

3. copy DDL（ALGORITHM=COPY）：如"修改列类型"、"修改表字符集"操作，只支持 table copy，会阻塞 DML 操作，不属于 Online DDL。对此类 DDL 语句，建议使用第三方在线改表工具 ptosc 完成。

总体来看，传统单机 MySQL 除了 instant DDL 外，主流仍是采用第三方在线改表工具来执行 DDL 操作。然而，这种方法有一些明显的缺点：

1. DDL 可能因大型事务或长查询而无法获得锁，导致持续等待和重复失败。
2. 所有第三方工具都需要重建整个表，为了确保稳定性而大幅牺牲性能。经测试表明，与原生 DDL 执行模式（INSTANT / INPLACE / COPY）相比，第三方工具执行 DDL 的性能下降了10倍甚至几个数量级，考虑到当前不断快速增长的数据量，这是难以接受的。

相比传统单机 MySQL ，在分布式数据库中执行 DDL 将面临更多、更复杂的挑战：

1. 原生的 instant DDL 极速执行特性是否兼容和保留？
2. 如何解决需要数据回填的 DDL 同 DML 之间的并发控制问题，且同时兼顾执行效率？
3. 原生分布式数据库大多是存算分离架构，每个计算节点之间是弱关联关系（无状态），当在某一个计算节点上执行 DDL 变更的时候，同实例中其他计算节点如何及时感知这个 DDL 的变更？同时，要在 DDL 变更过程中保证不会出现数据读写错误。
4. 原生分布式数据库中，数据被分散在多个节点上，无论是节点规模还是节点存储容量都要比单机 MySQL 大，是否可以通过存储层直接操作 + 多机并行结合的方式来大大加快需重建表的 DDL 操作？
5. 分布式数据库 DDL 的 crash-safe 问题。

下面我们将探讨 TDStore 如何运用一系列创新的策略来克服 Online DDL 所面临的各种挑战。

1. 通过引入多版本 schema 解决 instant DDL 问题：

在表结构中引入版本号（schema version）概念。如上图中的 t1 表，初始版本为1，执行加列操作后，版本号变为2。新数据行在插入时会附着版本号。读取数据时需要先判断数据行的版本，如果数据行版本为2，就用当前的表结构进行解析；如果数据行版本为1，比当前版本小，确定 F2 列不在该版本的 schema 中后，直接填充默认值再返回到客户端。通过引入上述多版本 schema 解析规则，使得 Add column，varchar 扩展长度等无损类型变更只需修改元数据的 DDL 瞬间完成。

2. 通过引入托马斯写入法则（Thomas Write Rule）解决需数据回填的 DDL 同 DML 之间的并发控制问题：

通过引入托马斯写入法则（Thomas Write Rule）为 DDL & DML 操作提供并发控制机制，保证了数据库协议的序列化顺序。相比标准的时间戳并行控制方法，托马斯写入法则（Thomas Write Rule）忽略过期写，减少了交易被撤回的几率。

3. 参考 Google F1 的 schema 变更方法，引入过渡态，确保 schema 变更过程中是平滑的：

Google F1 的 schema 变更方法参考 Online, Asynchronous Schema Change in F1 。

从中可以抽象出的基本概念：当一件事情无法立即从一个状态（添加索引前）转变为另一个状态（添加索引后）时，可以尝试引入过渡性的中间状态，使相邻的两个状态兼容。这样尽管大家无法立刻进入某个状态，但只要身处兼容的两个状态下，并在不断往前转变状态的过程中依然保证所处状态的兼容性，就能按照这个模式逐步进入到最终状态，完成整个过渡。

上图以 DDL 中典型的 add index 举例，所有的 SQLEngine 从 v1 -> v4 状态变更过程中，引入了 v2、v3 两个过渡态，我们可以看到：

3.1 **v1 与 v2 共存**：v1 是 old schema，v2 是 new schema（delete-only）。此时所有的 SQLEngine 要么处于 v1 状态，要么处于 v2 状态。由于 index 只有在 delete 的时候才被操作，此时还没有 index 生成，不会有数据不一致的问题。

3.2 **v2 与 v3 共存**：v2 是 new schema（delete-only），v3 是 new schema（write-only）。此时所有的 SQLEngine 要么处于 v2 状态，要么处于 v3 状态。处于 v2 状态的 SQLEngine 可以正常插入数据、删除数据和索引（存量和新增数据都缺失索引）；处于 v3 状态的 SQLEngine 可以正常插入和删除数据和索引。但是由于 v2 和 v3 状态下，索引对于用户都是不可见的，用户能看到的只有数据，所以对用户而言还是满足数据一致性的。

3.3 **v3 与 v4 共存**：v3 是 new schema（write-only），v4 是 new schema（添加索引完成）。此时所有的 SQLEngine 要么处于 v3 状态，要么处于 v4 状态。处于 v3 状态的 SQLEngine 可以正常插入和删除数据和索引（存量数据仍缺失索引）；处于 v4 状态的 SQLEngine 已将存量数据的索引补全，是 add index 后的最终形态。 可以发现 v3 状态下，用户能看见完整的数据；v4 状态下，完整的数据和索引均能看见，因此数据也是一致的。

4. 结合过渡态思想，设计 Write Fence 机制：通过在存储层对请求做版本校验，保证了在任意时刻多个 SQLEngine 的有效写入只能在两个相邻的状态之间，不会产生数据不一致。

Write Fence 是 TDStore 的内部数据结构，存储的是 schema_obj_id → schema_obj_version 的映射。Write Fence 解决的是计算层 SQLEngine 与存储层 TDStore 联动的问题，我们需要在推进 SQLEngine 状态前让 TDStore 感知到相关情况。

每个 SQLEngine 在进入下一状态前，需将当前版本推送至 TDStore 保存（push write fence）。只要 push 成功则保证两点约束：

- 当前系统中小于推送版本的请求已全部完成。
- 后续如果出现小于推送版本的请求会被拒绝。

通过存储层的版本校验机制，保证了在任意时刻多个 SQLEngine 的有效写入只能在两个相邻的状态之间，不会产生数据不一致。
上图中同时展示了即使在极端异常的场景中，假设某一节点在 push v2 版本已经成功的情况下，仍发送小于 v2 的请求，这时存储层 TDStore 就会发现该请求比当前 Write Fence 中的 v2 要小，从而拒绝执行，保证了整体算法运行的正确性。

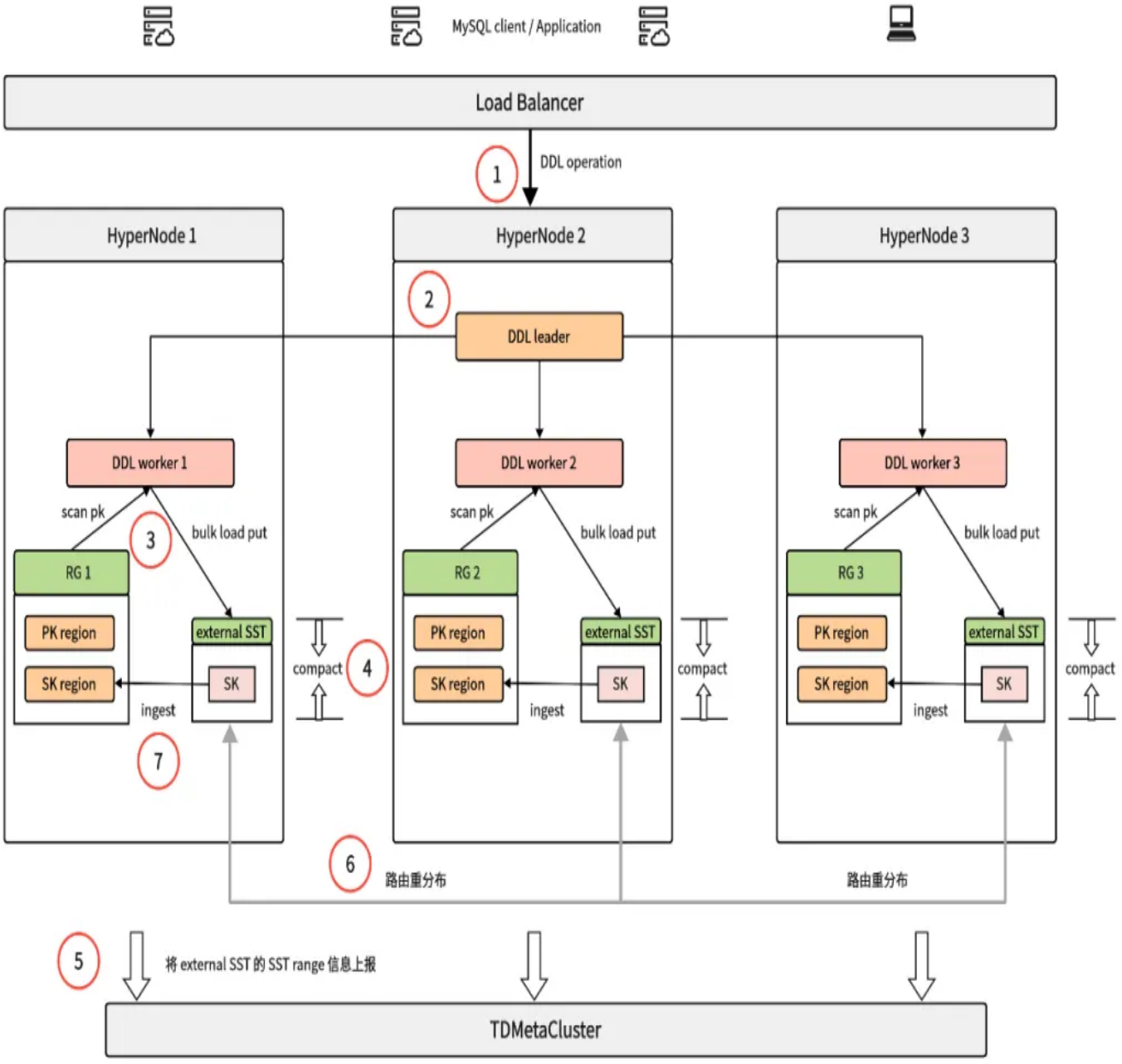
5. DDL 的 crash-safe 问题：

引入 DDL 任务队列 + DDL 恢复线程，保证 DDL 的 crash-safe。需要注意的是，恢复线程是独立于工作线程之外的，彼此之间不会形成等待通信。如果由于网络原因导致执行失败，恢复线程会执行接管操作（恢复线程不一定与原工作线程位于同一个节点）。监控 DDL 任务队列的方式也非常简单，可查询 `INFORMATION_SCHEMA.DDL_JOBS` 字典表，通过 `DDL_STATUS` 字段确认最终执行结果。
目前 TDStore 已经支持大多数场景下的 Online DDL 能力，详细请参考 OnlineDDL 说明 。

TDStore Fast Online DDL 新特性

从上面介绍我们可以看到，TDStore 如果在 Online DDL 中产生数据回填，采用的是托马斯写入法则（Thomas Write Rule）的方式进行批量回填，并通过并行来进行加速。但在实际使用中，特别是大数据量的场景中，由于 Thomas Write 需要对回填数据和用户并发的 DML 数据使用时间戳方法对比以确定保留哪个版本，导致执行的时

间依然比较长；因此最新版本的 TDStore，引入了 Fast Online DDL 功能：采用 bulk load 的方式，将回填数据组成 external sst，通过 ingest 的方式直接导入 TDStore 的 Lmax 层，省去了时间戳对比并发控制的开销，可以极大提升执行效率。



上图是 Fast Online DDL 的架构图，我们来看一下它的工作原理：

1. 用户发起 DDL 操作请求，通过负载均衡器随机发往 TDstore 实例中的对等节点，这里假设发往 HyperNode 2 节点。
2. 由 HyperNode 2 节点执行 DDL 操作，称为 DDL leader，它按照 DDL 所操作对象的 PK 数据分布将 worker 分配到所有相关的 HyperNode 对等节点。
3. 每个 worker scan 本地的 PK 生成索引数据，用 bulk load 方式写入 external sst 文件。

4. 所有 worker 完成 scan 所有的 PK 后，对 external sst 文件做一次压实操作（compact），保证同层 sst 文件之间 key 值不重复。
5. 将压实后的 sst range 信息上报给 TDMetaCluster（mc）管控节点，mc 根据这个信息将这些 range 划分 region（逻辑概念，一段段的小数据范围）并分配给对应的 RG（Replication Group，逻辑概念，一个 RG 包含多个 region，是最小的多副本数据同步单元），生成新的路由信息。同时 mc 暂时关闭 SK 涉及 region 路由的分裂合并。
6. HyperNode 根据新的路由信息重新组织 external sst 文件传输到对应的 HyperNode 节点。
7. 将满足新路由的 external sst 文件直接 ingest 到 user RG 的 sst 文件的 Lmax 层，完成后 mc 放开对 SK 涉及 region 路由的分裂合并限制。

TDStore Fast Online DDL 实践

以下步骤将创建一张大分区表，使用 add index 的 DDL 语句来测试 Fast Online DDL 在执行性能上的提升。

硬件环境

节点类型	节点规格	节点个数
HyperNode	16Core CPU/32GB Memory/增强型 SSD 云硬盘300GB	3

数据准备

```
-- 创建分区表，分区数为节点倍数：
CREATE TABLE `lineitem` (
  `L_ORDERKEY` int NOT NULL,
  `L_PARTKEY` int NOT NULL,
  `L_SUPPKEY` int NOT NULL,
  `L_LINENUMBER` int NOT NULL,
  `L_QUANTITY` decimal(15,2) NOT NULL,
  `L_EXTENDEDPRICE` decimal(15,2) NOT NULL,
  `L_DISCOUNT` decimal(15,2) NOT NULL,
  `L_TAX` decimal(15,2) NOT NULL,
  `L_RETURNFLAG` char(1) NOT NULL,
  `L_LINESTATUS` char(1) NOT NULL,
  `L_SHIPDATE` date NOT NULL,
  `L_COMMITDATE` date NOT NULL,
  `L_RECEIPTDATE` date NOT NULL,
  `L_SHIPINSTRUCT` char(25) NOT NULL,
  `L_SHIPMODE` char(10) NOT NULL,
  `L_COMMENT` varchar(44) NOT NULL,
  PRIMARY KEY (`L_ORDERKEY`,`L_LINENUMBER`)
) ENGINE=ROCKSDB DEFAULT CHARSET=utf8mb3
PARTITION BY HASH (`L_ORDERKEY`) PARTITIONS 24;
```

```
--造数可使用 TPC 官方标准工具：TPC-H v3.0.1，下载地址：tpc.org/tpch/

--导入，其中 LOAD DATA 的数据文件目录替换成自己的目录：
#!/bin/bash
for tbl in lineitem
do
  for i in {1..30}
  do
    echo "Importing table: $tbl"
    mysql $opts -e "set tdsql_bulk_load_allow_unsorted=1;set
tdsql_bulk_load = 1;LOAD DATA INFILE '/data/TPCH_test/dbgen/tpch-
100g/${tbl}.tbl.$i' INTO TABLE $tbl FIELDS TERMINATED BY '|';" &
  done
done
wait
date

--确认记录数：
sql> select count(*) from tpchpart100g.lineitem;
+-----------+
| count(*)  |
+-----------+
| 600037902 |

--预期数据大致均匀的分布在每个节点：
select sum(region_stats_approximate_size) as size, count(b.rep_group_id)
as region_nums, sql_addr, c.leader_node_name, b.rep_group_id from
information_schema.META_CLUSTER_DATA_OBJECTS a join
information_schema.META_CLUSTER_REGIONS b join
information_schema.META_CLUSTER_RGS c join
information_schema.META_CLUSTER_NODES d    on a.data_obj_id =
b.data_obj_id and b.rep_group_id = c.rep_group_id and c.leader_node_name
= d.node_name where a.table_name = 'lineitem' and a.data_obj_type =
'PARTITION_L1' group by rep_group_id order by leader_node_name;
+------------+-------------+------------------+----------------------+--
------------+
| size       | region_nums | sql_addr         | leader_node_name     |
rep_group_id |
+------------+-------------+------------------+----------------------+--
------------+
| 8879148586 |          76 | 9.30.0.133:15070 | node-three-001       |
64535 |
```

```
| 8878971995 |          81 | 9.30.2.175:15088 | node-three-002       |
513 |
| 8878089427 |          79 | 9.30.0.134:15070 | node-three-003       |
65082 |
+------------+-------------+------------------+----------------------+--
------------+

--若数据分布不均匀，则进行分裂和切主
ALTER INSTANCE SPLIT RG 64535 BY 'size';
ALTER INSTANCE TRANSFER LEADER RG 64535 TO 'node-three-003';
```

Fast Online DDL 开启前后对比

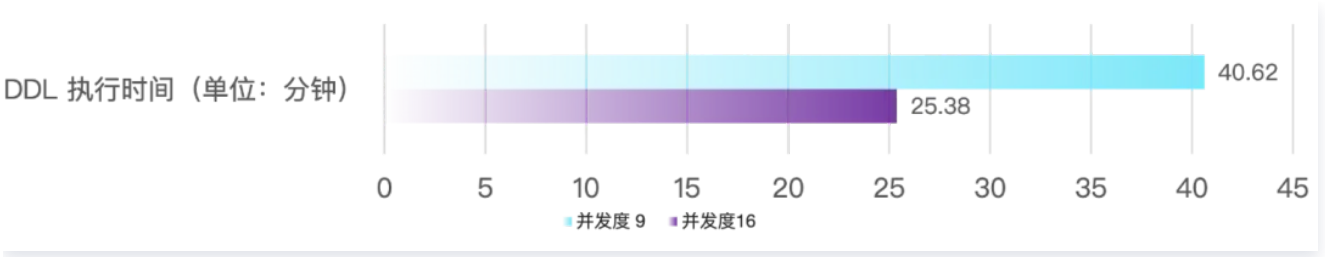
相关参数：

```
-- DDL 操作 worker 线程数（所有节点的 worker 总和），缺省值为 8：
max_parallel_ddl_degree

-- DDL 操作的数据回填模式，缺省值为'ThomasWrite'，要开启 Fast Online DDL 功能则
需要设置成'IngestBehind'：
tdsql_ddl_fillback_mode
```

使用默认的 'ThomasWrite' 数据回填模式【该模式下 DDL 线程为单机执行】，分别开启 9，16 线程数【不超单节点 CPU 数】测试 add index：

```
-- ThomasWrite 回填模式：
set session max_parallel_ddl_degree=9;
set session tdsql_ddl_fillback_mode='ThomasWrite';
alter table tpchpart100g.lineitem add index
index_idx_q_part_key(l_partkey);
Query OK, 0 rows affected (40 min 37.62 sec)

set session max_parallel_ddl_degree=16;
set session tdsql_ddl_fillback_mode='ThomasWrite';
alter table tpchpart100g.lineitem add index
index_idx_w_part_key(l_partkey);
Query OK, 0 rows affected (25 min 22.95 sec)
```

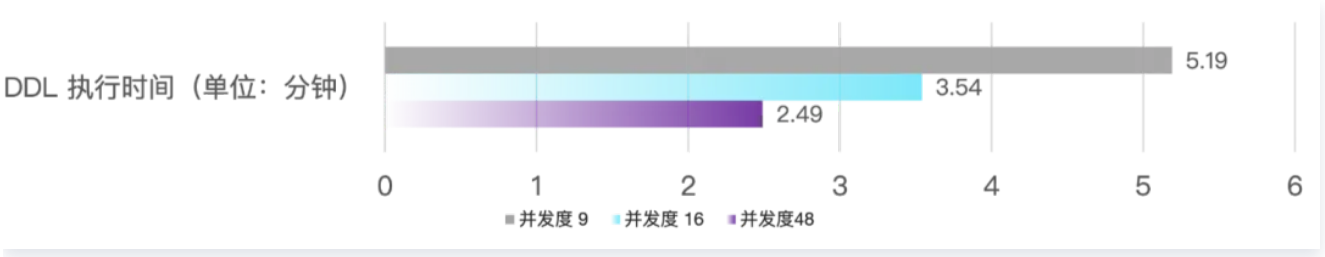
未开启 Fast Online DDL，执行时间对比：



开启 Fast Online DDL，使用 'IngestBehind' 数据回填模式【该模式下 DDL 线程为多机执行】，分别开启 9，16，48 线程数【不超数据分布节点 CPU 总数】测试 add index：

```
-- IngestBehind 回填模式：
set session max_parallel_ddl_degree=9;
set session tdsql_ddl_fillback_mode='IngestBehind';
alter table tpchpart100g.lineitem add index
index_idx_j_part_key(l_partkey);
Query OK, 0 rows affected (5 min 11.66 sec)

set session max_parallel_ddl_degree=16;
set session tdsql_ddl_fillback_mode='IngestBehind';
alter table tpchpart100g.lineitem add index
index_idx_k_part_key(l_partkey);
Query OK, 0 rows affected (3 min 32.15 sec)

set session max_parallel_ddl_degree=48;
set session tdsql_ddl_fillback_mode='IngestBehind';
alter table tpchpart100g.lineitem add index
index_idx_l_part_key(l_partkey);
Query OK, 0 rows affected (2 min 29.52 sec)
```

开启 Fast Online DDL，执行时间对比：



DDL 执行结果查询，DDL_STATUS 字段显示最终结果：

```
--ddl_jobs：记录ddl执行流程的字典表
select * from INFORMATION_SCHEMA.DDL_JOBS where
date_format(START_TIMESTAMP,'%Y-%m-%d')='2024-11-22' and IS_HISTORY=1
```

```
order by START_TIMESTAMP desc limit 1\G
*************************** 1. row ***************************
             ID: 18
    SCHEMA_NAME: tpch100g
     TABLE_NAME: lineitem
        VERSION: 204
     DDL_STATUS: SUCCESS
START_TIMESTAMP: 2024-11-22 18:47:47
 LAST_TIMESTAMP: 2024-11-22 18:50:18
        DDL_SQL: alter table tpch100g.lineitem add index
index_idx_s_part_key(l_partkey)
      INFO_TYPE: ALTER TABLE
           INFO: {"tmp_tbl":{"db":"tpch100g","table":"#sql-
11746_212c8b_673ef509000030_9"},"alt_type":1,"alt_tid_upd":
{"tid_from":10013,"tid_to":10013},"cr_idx":
[{"id":10240,"ver":34,"stat":0,"tbl_type":2,"idx_type":2},
{"id":10241,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10242,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10243,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10244,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10245,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10246,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10247,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10248,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10249,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10250,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10251,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10252,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10253,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10254,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10255,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10256,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10257,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10258,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10259,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10260,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10261,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10262,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10263,"ver":34,"stat":0,"tbl_type":2,"idx_type":4},
{"id":10264,"ver":34,"stat":0,"tbl_type":2,"idx_type":4}],"rm_idx":
[],"init":false,"tmp_tab":false,"online_op":true,"wf_rmed":false,"online
_copy_stage":0,"idx_op":true,"row_applied":true,"row_apply_saved":true,"
current_schema_name":"tpch100g","crt_data_obj_task_id":29062709316158283
```

```
,"dstr_data_obj_task_id":0,"data_obj_to_be_dstr_arr":
[],"part_policy_ids":[1],"progress":"total: 200, scanned: 200
(100.00%)","fillback_mode":"IngestBehind","exec_addr":
{"ip":"10.0.20.144","port":6008},"recov_addr":
{"ip":"10.0.20.144","port":6008}}
     IS_HISTORY: 1
1 row in set (0.00 sec)
```

ddl_jobs 字段详解：

字段	说明
ID	每个 DDL JOB 都有唯一 ID。
SCHEMA_NAME	库名。
TABLE_NAME	表名。
VERSION	INFO 字段解析版本号。
DDL_STATUS	DDL JOB 执行状态，有 SUCCESS，FAIL，EXECUTING 三种状态。
START_TIMESTAMP	DDL JOB 发起时间。
LAST_TIMESTAMP	DDL JOB 结束时间。
DDL_SQL	DDL 语句明细。
INFO_TYPE	DDL 语句类型。
INFO	执行 DDL 的执行过程中的元数据信息（包含 add index, copy table 类型 DDL 语句的执行进度）。

TDStore Online DDL 的使用建议

TDStore 的 Fast Online DDL 能力，通过并行处理和旁路写入相结合，使得 DDL 操作变得更加高效和便捷。但如果我们没有正确地划分大/小表，或者没有根据数据规模进行适当的分区，那么 Fast Online DDL 的执行效率可能会大打折扣。这是因为一张大表，在没有进行适当分区的情况下，数据很可能都集中在单个节点上，因此 DDL 操作也会在单个节点上进行，而不是在多个节点上并行执行，这将大大降低执行效率。

只有根据数据规模合理的使用分区表，才能充分运用 Fast Online DDL 的分布式可扩展性能。

创建分区建议：

1. TDStore 100% 兼容原生 MySQL 分区表语法，支持一/二级分区，主要用于解决：（1）大表的容量问题；（2）高并发访问的性能问题。
2. 大表的容量问题：如果单表大小预期未来将超过实例单节点数据盘大小，建议创建一级 hash 或 key 分区将数据均匀打散到多个节点上；如果未来数据量再增大，可通过弹性扩容的方式不断“打薄”磁盘水位。
3. 高并发访问的性能问题：高并发访问 TP 业务，如果预计单节点性能无法扛住超量读写压力的时候，也建议创建一级 hash 或 key 分区将读写压力均匀打散到多个节点上。
4. 第2、第3点中创建的分区表，建议结合业务特点选择能满足大部分核心业务查询的字段作为分区键，分区数建议为实例节点数量的倍数。
5. 如果有数据清理的需求，可创建 RANGE 分区表使用 truncate partition 命令进行快速数据清理；如果要再兼顾数据打散，可进一步创建二级分区为 HASH 的分区表。

TDStore Online DDL 的发展

TDStore 正在飞速发展，各项功能立足于用户的需求正在快速迭代与完善。DDL 并行性能仍在持续优化中，目前 TDStore 的新版本着重优化了分区表在 add index 时的数据回填性能。因为我们观察到已上线的业务系统中，不少都是数 T、数十 T 的大分区表，且有需要在线添加索引的需求，因此这块能力我们进行了优先适配。在即将到来的下一个 TDStore 版本中，我们会将 Fast Online DDL 能力进行完整呈现，带来分区表在 copy table，以及普通表在 add index、copy table 时的 Fast Online DDL 能力。
作为腾讯云数据库长期战略的核心，TDStore 将持续以业务需求为导向，专注打磨产品，为用户提供更高效、更稳定的服务。