TiDB 测试

一、全量同步测试

使用 checker 进行 Schema 检查

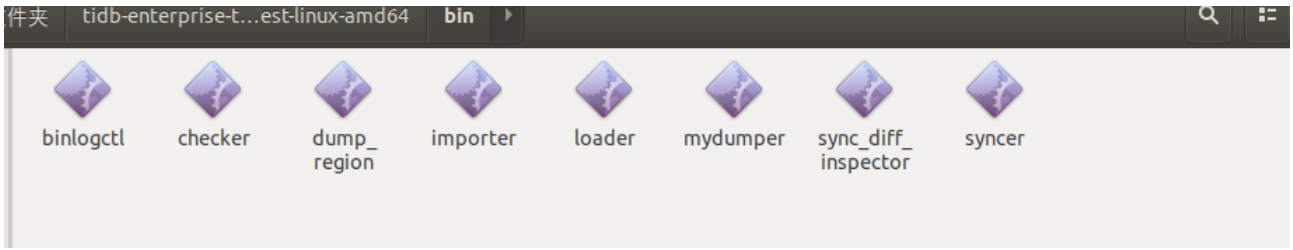
在迁移之前，我们可以使用 TiDB 的 checker 工具，checker 是我们开发的一个小工具，用于检测目标 MySQL 库中的表的表结构是否支持无缝的迁移到 TiDB，TiDB 支持绝大多数的 MySQL 常用的原生数据类型，所以大多数情况 checker 的返回应该是 ok。如果 check 某个 table schema 失败，表明 TiDB 当前并不支持，我们不能对该 table 里面的数据进行迁移。checker 包含在 TiDB 工具集里面，我们可以直接下载。

下载 TiDB 工具集

Linux

```
# 下载 tool 压缩包
wget http://download.pingcap.org/tidb-enterprise-tools-latest-linux-amd64.tar.gz
wget http://download.pingcap.org/tidb-enterprise-tools-latest-linux-amd64.sha256


# 检查文件完整性，返回 ok 则正确
sha256sum -c tidb-enterprise-tools-latest-linux-amd64.sha256
# 解开压缩包
tar -xzf tidb-enterprise-tools-latest-linux-amd64.tar.gz
cd tidb-enterprise-tools-latest-linux-amd64
```

解压后，可以看到在 bin 文件夹下有许多工具



使用 checker 检查的一个示范

- 在 MySQL 的 test database 里面创建几张表，并插入数据:

```
USE test;
CREATE TABLE t1 (id INT, age INT, PRIMARY KEY(id)) ENGINE=InnoDB;
CREATE TABLE t2 (id INT, name VARCHAR(256), PRIMARY KEY(id)) ENGINE=InnoDB;


INSERT INTO t1 VALUES (1, 1), (2, 2), (3, 3);
INSERT INTO t2 VALUES (1, "a"), (2, "b"), (3, "c");
```

- 使用 checker 检查 test database 里面所有的 table

```
./bin/checker -host 127.0.0.1 -port 3306 -user root test
2016/10/27 13:11:49 checker.go:48: [info] Checking database test
2016/10/27 13:11:49 main.go:37: [info] Database DSN:
root:@tcp(127.0.0.1:3306)/test?charset=utf8
2016/10/27 13:11:49 checker.go:63: [info] Checking table t1
2016/10/27 13:11:49 checker.go:69: [info] Check table t1 succ
2016/10/27 13:11:49 checker.go:63: [info] Checking table t2
2016/10/27 13:11:49 checker.go:69: [info] Check table t2 succ
```

- 使用 `checker` 检查 test database 里面某一个 table

这里，假设我们只需要迁移 table `t1`。

```
./bin/checker -host 127.0.0.1 -port 3306 -user root test t1
2016/10/27 13:13:56 checker.go:48: [info] Checking database test
2016/10/27 13:13:56 main.go:37: [info] Database DSN:
root:@tcp(127.0.0.1:3306)/test?charset=utf8
2016/10/27 13:13:56 checker.go:63: [info] Checking table t1
2016/10/27 13:13:56 checker.go:69: [info] Check table t1 succ
Check database succ!
```

一个无法迁移的 table 例子

我们在 MySQL 里面创建如下表：

```
CREATE TABLE t_error (
  c timestamp(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON UPDATE
CURRENT_TIMESTAMP(3)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

使用 `checker` 进行检查，会报错，表明我们没法迁移 `t_error` 这张表。

```
./bin/checker -host 127.0.0.1 -port 3306 -user root test t_error
2016/10/27 13:19:28 checker.go:48: [info] Checking database test
2016/10/27 13:19:28 main.go:37: [info] Database DSN:
root:@tcp(127.0.0.1:3306)/test?charset=utf8
2016/10/27 13:19:28 checker.go:63: [info] Checking table t_error
2016/10/27 13:19:28 checker.go:67: [error] Check table t_error failed with err:
line 1 column 56 near ") ON UPDATE CURRENT_TIMESTAMP(3)
) ENGINE=InnoDB DEFAULT CHARSET=latin1"
github.com/pingcap/tidb/parser/yy_parser.go:111:
github.com/pingcap/tidb/parser/yy_parser.go:124:
/home/jenkins/workspace/WORKFLOW_TOOLS_BUILDING/go/src/github.com/pingcap/tidb-
tools/checker/checker.go:122:  parse CREATE TABLE `t_error` (
  `c` timestamp(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON UPDATE
CURRENT_TIMESTAMP(3)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 error
/home/jenkins/workspace/WORKFLOW_TOOLS_BUILDING/go/src/github.com/pingcap/tidb-
tools/checker/checker.go:114:
2016/10/27 13:19:28 main.go:68: [error] Check database test with 1 errors and 0
warnings.
```

使用 mydumper/loader 全量导入数据

我们使用 `mydumper` 从 MySQL 导出数据，然后用 `loader` 将其导入到 TiDB 里面。

注意，虽然我们也支持使用 MySQL 官方的 `mysqldump` 工具来进行数据的迁移工作，但相比于 `mydumper/loader`，性能会慢很多，对于大量数据的迁移会花费很多时间，这里我们并不推荐。

mydumper/loader 是一个更强大的数据迁移工具，具体可以参考 https://github.com/maxbube/mydumper。

从 MySQL 导出数据

我们使用 mydumper 从 MySQL 导出数据，如下:

```
./bin/mydumper -h 127.0.0.1 -P 3306 -u root -p 123456 -t 16 -F 128 -B test -T t1,t2 -o ./var/test
```

上面，我们使用 -B test 表明是对 test 这个 database 操作，然后用 -T t1,t2 表明只导出 t1，t2 两张表。

-t 16 表明使用 16 个线程去导出数据。-F 128 是将实际的 table 切分成多大的 chunk，这里就是 128MB 一个 chunk。

注意：在阿里云一些需要 super privilege 的云上面，mydumper 需要加上 --no-locks 参数，否则会提示没有权限操作。

给 TiDB 导入数据

我们使用 loader 将之前导出的数据导入到 TiDB。

```
./bin/loader -h 127.0.0.1 -P 4000 -u root -p 123456 -t 16 -d ./var/test
```

导入成功之后，我们可以用 MySQL 官方客户端进入 TiDB，查看:

```
mysql -h 127.0.0.1 -P 4000 -u root

mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
| t2             |
+----------------+

mysql> select * from t1;
+----+------+
| id | age  |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
+----+------+

mysql> select * from t2;
+----+------+
| id | name |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c
```

全量同步测试结果

（1）导入开始时间

```
2018/09/26 12:36:37 loader.go:154: [info] [loader][restore table data sql]/home/
niuwenju/tidb-enterprise-tools-latest-linux-amd64/data/test_TiDB.test.sql[start]
```

（2）导入结束时间

```
2018/09/26 12:38:33 status.go:32: [info] [loader] finished_bytes = 40890358, tot
al_bytes = GetAllRestoringFiles40890058, progress = 100.00 %
```

（3）全量同步用时

开始时间	结束时间	用时	数据大小	数据量
12:36:37	12:38:33	01:56	121.69M	150 万条

二、使用 syncer 增量导入数据实现数据和 MySQL 实时同步

TiDB 提供 `syncer` 工具能方便的将 MySQL 的数据增量的导入到 TiDB 里面。

MySQL 开启 binlog

在使用 `syncer` 之前，我们必须保证：

- MySQL 开启 binlog 功能

  1、开启 mysql 配置文件中的 server-id、log_bin 这两个配置项

  ```
  server-id               = 1
  log_bin                 = /var/log/mysql/mysql-bin.log
  ```

  2、使用命令 **service mysql restart** 重启 mysql 服务
- Binlog 格式必须使用 `row` format，这也是 MySQL 5.7 之后推荐的 binlog 格式，可以使用如下语句打开:

```
    SET GLOBAL binlog_format = ROW;
```

获取同步 position

我们通过 `show master status` 得到当前 binlog 的 position，`syncer` 的初始同步位置就是从这个地方开始。

```
show master status;
+------------------+----------+--------------+------------------
+-------------------+
```

```
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
Executed_Gtid_Set |
+------------------+----------+--------------+------------------
+-------------------+
| mysql-bin.000003 |     1280 |              |                  |
|
+------------------+----------+--------------+------------------
+-------------------+
```

我们将 position 相关的信息保存到一个 `syncer.meta` 文件里面，用于 `syncer` 的同步:

```
# cat syncer.meta
binlog-name = "mysql-bin.000003"
binlog-pos = 1280
```

注意：`syncer.meta` 只需要第一次使用的时候配置，后续 `syncer` 同步新的 binlog 之后会自动将其更新到最新的 position。

启动 syncer

`syncer` 的配置文件 `config.toml`:

```
log-level = "info"

server-id = 101

# meta 文件地址
meta = "./syncer.meta"
worker-count = 16
batch = 1

pprof-addr = ":10081"

[from]
host = "127.0.0.1"
user = "root"
password = ""
port = 3306

[to]
host = "127.0.0.1"
user = "root"
password = ""
port = 4000
```

上图中 from 为M ysql 的信息，to 为T iDB 的信息。

启动 `syncer`:

```
./bin/syncer -config config.toml

2016/10/27 15:22:01 binlogsyncer.go:226: [info] begin to sync binlog from
position (mysql-bin.000003, 1280)
2016/10/27 15:22:01 binlogsyncer.go:130: [info] register slave for master server
127.0.0.1:3306
2016/10/27 15:22:01 binlogsyncer.go:552: [info] rotate to (mysql-bin.000003,
1280)
2016/10/27 15:22:01 syncer.go:549: [info] rotate binlog to (mysql-bin.000003,
1280)
```

创建数据添加工程

(1)创建 python 工程

(2)编写数据添加程序

1)添加表

```
conn = MySQLdb.connect(host='127.0.0.1', user='root', passwd='123456',
port=3306)
conn.autocommit(True)
cur = conn.cursor()
cur.execute("use test_TiDB;create table test(id VARCHAR(40) NOT
NULL,time VARCHAR(256) NOT NULL,PRIMARY KEY ( id ))ENGINE=InnoDB
DEFAULT CHARSET=utf8;")
cur.close()
```

2)单线程添加数据(一次一条)

```
conn = MySQLdb.connect(host='127.0.0.1', user='root', passwd='123456',
port=3306)
```

```
conn.autocommit(True)
number = 500000
for i in range(number):
    cur = conn.cursor()
    ct = time.time()
    local_time = time.localtime(ct)
    data_head = time.strftime("%H:%M:%S", local_time)
    data_secs = (ct - int(ct)) * 1000
    time_stamp = "%s.%03d" % (data_head, data_secs)
    sql = 'use test_TiDB;INSERT INTO test(id,time) VALUES (' + str(i) +
',' + "\'" + time_stamp + "\'" + ');'
    try:
        cur.execute(sql)
        cur.close()
        conn.commit()
    except:
        # Rollback in case there is any error
        conn.rollback()
```

3)单线程添加数据（一次十条）

```
conn = MySQLdb.connect(host='127.0.0.1', user='root', passwd='123456',
port=3306)
```

```
conn.autocommit(True)
number = 50000
for i in range(number):
    cur = conn.cursor()
    ct = time.time()
    local_time = time.localtime(ct)
    data_head = time.strftime("%H:%M:%S", local_time)
```

```
    data_secs = (ct - int(ct)) * 1000
    time_stamp = "%s.%03d" % (data_head, data_secs)
    sql = 'use test_TiDB;INSERT INTO test(id,time) VALUES (' +
str(i*10) + ',' + "\'" + time_stamp + "\'" + '),(' + str(i*10+1) + ','
+ "\'" + time_stamp + "\'" + '),(' + str(i*10+2) + ',' + "\'" +
time_stamp + "\'" + '),(' + str(i*10+3) + ',' + "\'" + time_stamp +
"\'" + '),(' + str(i*10+4) + ',' + "\'" + time_stamp + "\'" + '),(' +
str(i*10+5) + ',' + "\'" + time_stamp + "\'" + '),(' + str(i*10+6) +
',' + "\'" + time_stamp + "\'" + '),(' + str(i*10+7) + ',' + "\'" +
time_stamp + "\'" + '),(' + str(i*10+8) + ',' + "\'" + time_stamp +
"\'" + '),(' + str(i*10+9) + ',' + "\'" + time_stamp + "\'" + ');'
    try:
        cur.execute(sql)
        cur.close()
        conn.commit()
    except:
        # Rollback in case there is any error
        conn.rollback()
```

4)多线程添加数据(一次一条)

```
def do(id,num):
```

```
    conn = MySQLdb.connect(host='127.0.0.1', user='root',
passwd='123456', port=3306)
    conn.autocommit(True)
    a = range(num)
    for i in a[id*len(a)/10:(id+1)*len(a)/10]:
        cur = conn.cursor()
        ct = time.time()
        local_time = time.localtime(ct)
        data_head = time.strftime("%H:%M:%S", local_time)
        data_secs = (ct - int(ct)) * 1000
        time_stamp = "%s.%03d" % (data_head, data_secs)
        sql = 'use test_TiDB;INSERT INTO test(id,time) VALUES (' +
str(i) + ',' + "\'" + str(time_stamp) + "\'" + ');'
        try:
            cur.execute(sql)
            cur.close()
            conn.commit()
        except:
            # Rollback in case there is any error
            conn.rollback()
thread = 10
number = 100000
for i in range(thread):
    t = threading.Thread(target=do,args=(i,number))
    t.start()
```

5)多线程添加数据(一次十条)

```
def do(id,num):
```

```
    conn = MySQLdb.connect(host='127.0.0.1', user='root',
passwd='123456', port=3306)
    conn.autocommit(True)
    a = range(num)
    for i in a[id*len(a)/10:(id+1)*len(a)/10]:
        cur = conn.cursor()
        ct = time.time()
        local_time = time.localtime(ct)
        data_head = time.strftime("%H:%M:%S", local_time)
        data_secs = (ct - int(ct)) * 1000
        time_stamp = "%s.%03d" % (data_head, data_secs)
        sql = 'use test_TiDB;INSERT INTO test(id,time) VALUES (' +
str(i*10) + ',' + "\'" + time_stamp + "\'" + '),(' + str(i*10+1) + ','
+ "\'" + time_stamp + "\'" + '),(' + str(i*10+2) + ',' + "\'" +
time_stamp + "\'" + '),(' + str(i*10+3) + ',' + "\'" + time_stamp +
"\'" + '),(' + str(i*10+4) + ',' + "\'" + time_stamp + "\'" + '),(' +
str(i*10+5) + ',' + "\'" + time_stamp + "\'" + '),(' + str(i*10+6) +
',' + "\'" + time_stamp + "\'" + '),(' + str(i*10+7) + ',' + "\'" +
time_stamp + "\'" + '),(' + str(i*10+8) + ',' + "\'" + time_stamp +
"\'" + '),(' + str(i*10+9) + ',' + "\'" + time_stamp + "\'" + ');'
        try:
            cur.execute(sql)
            cur.close()
            conn.commit()
        except:
            # Rollback in case there is any error
            conn.rollback()
thread = 10
number = 10000
for i in range(thread):
    t = threading.Thread(target=do,args=(i,number))
    t.start()
```

(3)编写数据查询程序

```
def test():
    conn = MySQLdb.connect(host='10.0.5.107', user='root',
passwd='its7888$', port=4000)
    conn.autocommit(True)
    timee = None
    while timee == None:
        cur = conn.cursor()
        if cur.execute("use test_TiDB;select * from test where id =
499999;") == 0:
            timee = None
        else:
            # print(1)
            ct = time.time()
            local_time = time.localtime(ct)
            data_head = time.strftime("%H:%M:%S", local_time)
            data_secs = (ct - int(ct)) * 1000
            timee = "%s.%03d" % (data_head, data_secs)
```

```
        cur.close()
    conn.commit()
    return timee
print(test())
```

增量同步具体测试过程

（1）单线程测试

1)根据测试数据大小，更改查询语句的 id 值，然后启动程序

```
if cur.execute("use test_TiDB;select * from test where id = 499999;") == 0:
    timee = None
```

2)根据测试数据大小，更改 number 大小，启动程序

```
number = 500000
for i in range(number):
```

3)查看 syncer 启动窗口，确定同步开始时间

```
2018/09/25 17:53:48 meta.go:137: [info] save position to file, binlog-name:mysql
-bin.000025 binlog-pos:40169269 binlog-gtid:
```

4)查看查询程序输出结果，确定同步结束时间

```
/home/niuwenju/Desktop/testTiDB/venv
17:47:05.680

Process finished with exit code 0
```

5)使用 SQL 语句： select * from test where id=0; 确定数据添加开始时间

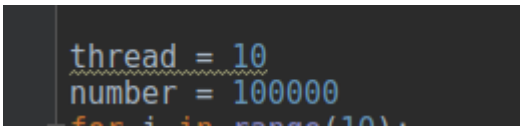
```
+----+--------------+
| id | time         |
+----+--------------+
| 0  | 17:53:48.231 |
```

6)使用 SQL 语句： select * from test where id=？；确定数据添加结束时间（？处的值与查询程序中的 id 值相同）

```
+-------+--------------+
| id    | time         |
+-------+--------------+
| 99999 | 18:01:59.191 |
+-------+--------------+
```

（2）多线程测试

同单线程，只是在设置 number 值时，同时设置 thread 值,一次插入一条与十条的 number 值一样。假设要添加十万条数据，可以设置 thread 为 10,number 为 10000



增量同步测试结果

(1)单线程添加数据

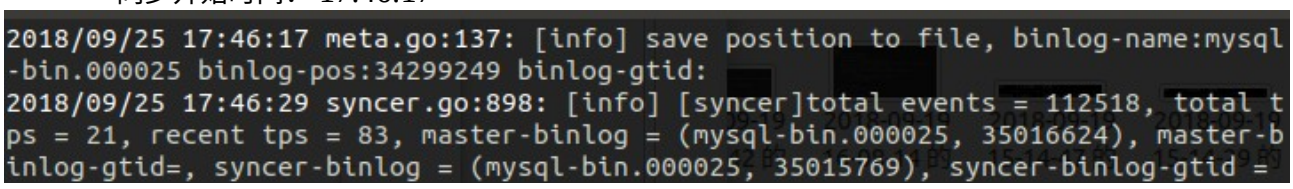
测试代码：（python）

1、同步添加一万条数据（一次插入 1 个）

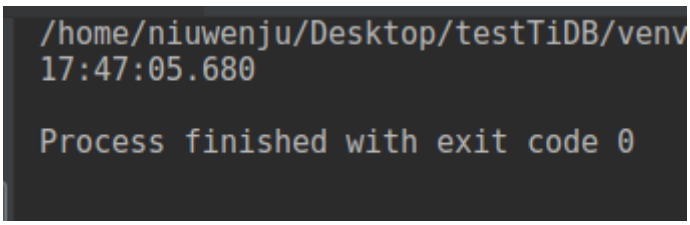
添加开始时间：17:46:17.923

添加结束时间：17:47:05.581

同步开始时间： 17:46:17

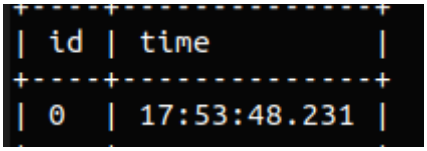


同步结束时间： 17:47:05.680



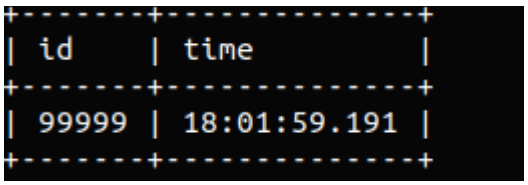
2、同步添加十万条数据（一次插入 1 个）

添加开始时间: 17:53:48.231



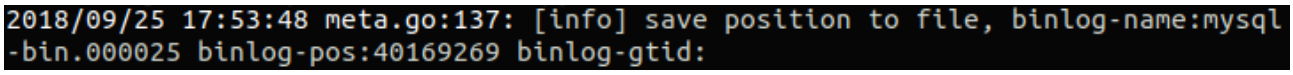
```
+----+--------------+
| id | time         |
+----+--------------+
| 0  | 17:53:48.231 |
+----+--------------+
```

添加结束时间： 18:01:59.191



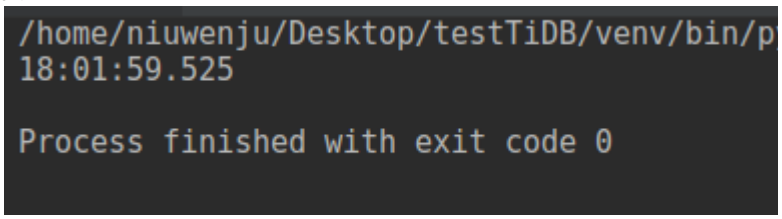
```
+-------+--------------+
| id    | time         |
+-------+--------------+
| 99999 | 18:01:59.191 |
+-------+--------------+
```

同步开始时间： 17:53:48



```
2018/09/25 17:53:48 meta.go:137: [info] save position to file, binlog-name:mysql
-bin.000025 binlog-pos:40169269 binlog-gtid:
```

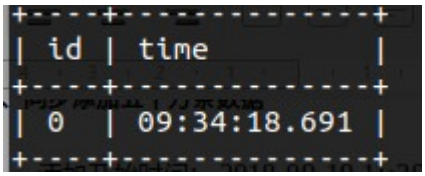
同步结束时间： 18:01:59.525



```
/home/niuwenju/Desktop/testTiDB/venv/bin/p
18:01:59.525

Process finished with exit code 0
```

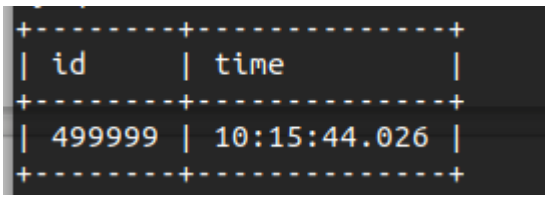
3、同步添加五十万条数据（一次插入 1 个）

添加开始时间： 09:34:18.691



```
+----+--------------+
| id | time         |
+----+--------------+
| 0  | 09:34:18.691 |
+----+--------------+
```

添加结束时间： 10:15:44.026



```
+--------+--------------+
| id     | time         |
+--------+--------------+
| 499999 | 10:15:44.026 |
+--------+--------------+
```

同步开始时间： 09:34:18

```
2018/09/26 09:34:18 meta.go:137: [info] save position to file, binlog-name:mysql
-bin.000027 binlog-pos:547263 binlog-gtid:
```

同步结束时间： 10:15:46.315

```
/home/niuwenju/Desktop/t
10:15:46.315
```

4、同步添加十万条数据(一次插入 10 个)

添加开始时间： 11:09:56.327

id	time
0	11:09:56.327

添加结束时间： 11:10:53.048

id	time
99999	11:10:53.048

同步开始时间： 11:09:56.327

同步结束时间： 11:12:22.999

```
/home/niuwenju/Desktop/testTiDB/venv/
11:12:22.999

Process finished with exit code 0
```

5、同步添加五十万条数据（一次插入 10 个）

添加开始时间：12:38:07.426

id	time
0	12:38:07.426

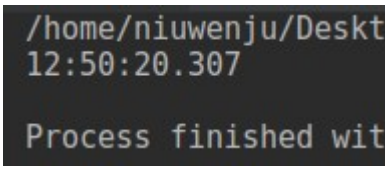
添加结束时间：12:43:00.778

id	time
499999	12:43:00.778

1 row in set (0.14 sec)

同步开始时间： 12:38:07

同步结束时间：12:50:20.307



6、小结

单线程增量同步统计表：

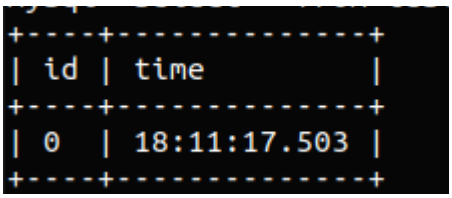
数据量	添加用时	同步用时
一万（1 条）	47.558	47.657
十万（1 条）	08:10.960	08:11.294
五十万（1 条）	41:25.335	41:27.624
十万（10 条）	56.721	2:26.672
五十万（10 条）	04:53.352	18:12.881

（2)多线程添加数据

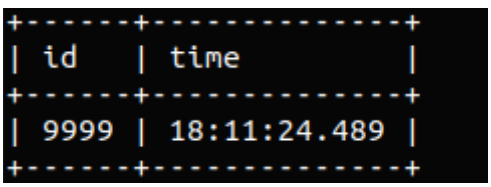
测试代码：（python）

1、添加 10 X 一千条数据(一次 1 条）

添加开始时间： 18:11:17.503



添加结束时间： 18:11:24.489



同步开始时间： 18:11:17

```
2018/09/25 18:11:17 meta.go:137: [info] save position to file, binlog-name:mysql
-bin.000025 binlog-pos:68758622 binlog-gtid:
```

同步结束时间： 18:11:37.624

```
/home/niuwenju/Desktop/testTiDB/venv/
18:11:37.624

Process finished with exit code 0
```

2、添加 10X 一万条数据（一次一条）

添加开始时间： 18:19:19.596

```
+----+--------------+
| id | time         |
+----+--------------+
| 0  | 18:19:19.596 |
+----+--------------+
```

添加结束时间： 18:20:30.616

```
+-------+--------------+
| id    | time         |
+-------+--------------+
| 99999 | 18:20:30.616 |
+-------+--------------+
```

同步开始时间： 18:19:19

```
2018/09/25 18:19:19 meta.go:137: [info] save position to file, binlog-name:mysql
-bin.000025 binlog-pos:71607975 binlog-gtid:
```

同步结束时间：18:22:41.034

```
/home/niuwenju/Desktop/testTiDB/venv/bin/pyt
18:22:41.034

Process finished with exit code 0
```

3、添加 10X 五万条数据（一次一条）

添加开始时间： 10:29:28.927

```
+----+--------------+
| id | time         |
+----+--------------+
| 0  | 10:29:28.927 |
+----+--------------+
```

添加结束时间： 10:35:29.775

```
+--------+--------------+
| id     | time         |
+--------+--------------+
| 499999 | 10:35:29.775 |
+--------+--------------+
```

同步开始时间： 10:29:29

```
2018/09/26 10:29:29 meta.go:137: [info] save position to file, binlog-name:mysql
-bin.000028 binlog-pos:39079111 binlog-gtid:
```

同步结束时间：10:46:22.889

```
/home/niuwenju/Desktop/testTiDB/venv
10:46:22.889

Process finished with exit code 0
```

4、添加 10X 一万条数据（一次 10 条）

添加开始时间：12:57:02.017

```
+----+--------------+
| id | time         |
+----+--------------+
| 0  | 12:57:02.017 |
+----+--------------+
```

添加结束时间：12:57:10.906

```
+-------+--------------+
| id    | time         |
+-------+--------------+
| 99999 | 12:57:10.906 |
+-------+--------------+
```

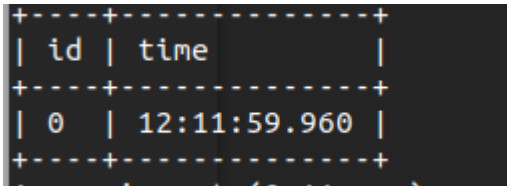
同步开始时间：12:57:02

同步结束时间：12:59:29.005

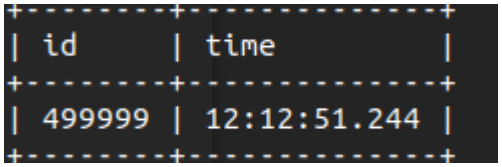
```
/home/niuwenju/Deskt
12:59:29.005

Process finished wit
```

5、添加 10X 五万条数据（一次 10 条）

添加开始时间：12:11:59.960

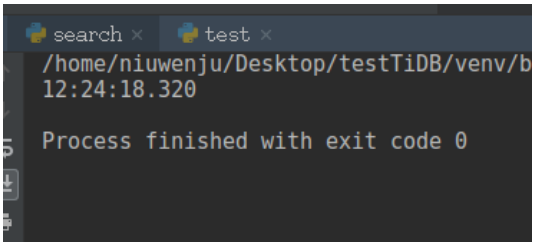


添加结束时间：12:12:51.224



同步开始时间：12:11:59

同步结束时间：12:24:18.320



6、小结

多线程增量同步统计表：

数据量	添加用时	同步用时
10 X 一千(1 条)	06.986	20.121
10 X 一万（1 条）	01:11.020	03:21.438
10 X 五万（1 条）	06:00.848	16:53.889
10X 一万（10 条）	08.889	02:26.988
10X 五万（10 条）	51.264	12:18.360

（3）单、多线程结果对比

单、多线程增量同步数据的比较：

数据总量	线程数	一次插入条数	添加用时	同步用时
10 万	1	1	08:10.960	08:11.294
	1	10	56.721	2:26.672
	10	1	01:11.020	03:21.438
	10	10	08.889	02:26.988

50 万	1	1	41:25.335	41:27.624
	1	10	04:53.352	18:12.881
	10	1	06:00.848	16:53.889
	10	10	51.264	12:18.360

三、TiDB 建表测试

(1)创建数据添加工程

(1)创建 python 工程

(2)编写表添加程序

1)单线程添加表

```
conn = MySQLdb.connect(host='10.0.5.107', user='root',
passwd='its7888$', port=4000)
conn.autocommit(True)
number = 10000
for i in range(number):
    cur = conn.cursor()
    ct = time.time()
    local_time = time.localtime(ct)
    data_head = time.strftime("%H_%M_%S", local_time)
    data_secs = (ct - int(ct)) * 1000
    time_stamp = "%s_%03d" % (data_head, data_secs)
    cur.execute('use testtable1;create table ' + 'time_' + str(i) + '_'
+ str(
        time_stamp) + '(tutorial_id INT NOT NULL AUTO_INCREMENT,PRIMARY
KEY ( tutorial_id ));')
    cur.close()
```

2)多线程添加表

```
def do(id,num):
    conn = MySQLdb.connect(host='10.0.5.107', user='root',
passwd='its7888$', port=4000)
    conn.autocommit(True)
    a = range(num)
    for i in a[id*len(a)/10:(id+1)*len(a)/10]:
        cur = conn.cursor()
        ct = time.time()
        local_time = time.localtime(ct)
        data_head = time.strftime("%H_%M_%S", local_time)
        data_secs = (ct - int(ct)) * 1000
        time_stamp = "%s_%03d" % (data_head, data_secs)
        cur.execute('use testtable1;create table ' + 'time_' + str(i) +
'_' + str(
            time_stamp) + '(tutorial_id INT NOT NULL
AUTO_INCREMENT,PRIMARY KEY ( tutorial_id ));')
        cur.close()
```

```
number = 10000
thread = 10
for i in range(thread):
    t = threading.Thread(target=do,args=(i,number))
    t.start()
```

(2)具体测试过程

同数据添加测试，不需要启动查询程序

(3)单线程建表

1、同步创建 5000 张表

创建开始时间：10:30:45:277

```
| time_0_10_30_45_277     |
```

创建结束时间：10:58:28:608

```
| time_4999_10_58_28_608 |
```

2、同步创建 1 万张表

创建开始时间：11:04:18:682

```
| time_0_11_04_18_682 |
```

创建结束时间：12:47:18:449

```
| time_9999_12_47_18_449 |
```

3、小结

单线程建表：

表的数量	用时
5000	27:43
1 万	1:43:00

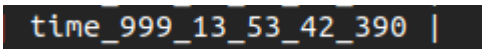
(4)多线程建表

1、10 线程创建 1000 张表

创建开始时间：13:48:40:363

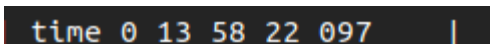
```
time_0_13_48_40_363
```

创建结束时间：13:53:42:390



2、10 线程创建 5000 张表

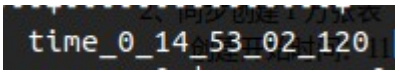
创建开始时间：13:58:22:097



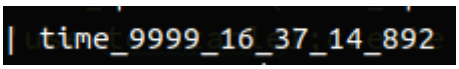
创建结束时间：14:34:20:494

3、10 线程创建 1 万张表

创建开始时间：14:53:02:120



创建结束时间：16:37:14:892



4、小结

多线程建表：

表的数量	用时
1000	5:02
5000	35:58
1 万	1:43:12

(5)单、多线程结果对比

单、多线程建表对比：

表的数量	线程数	用时
5000	1	27:43
	10	35:58
1 万	1	1:43:00
	10	1:43:12