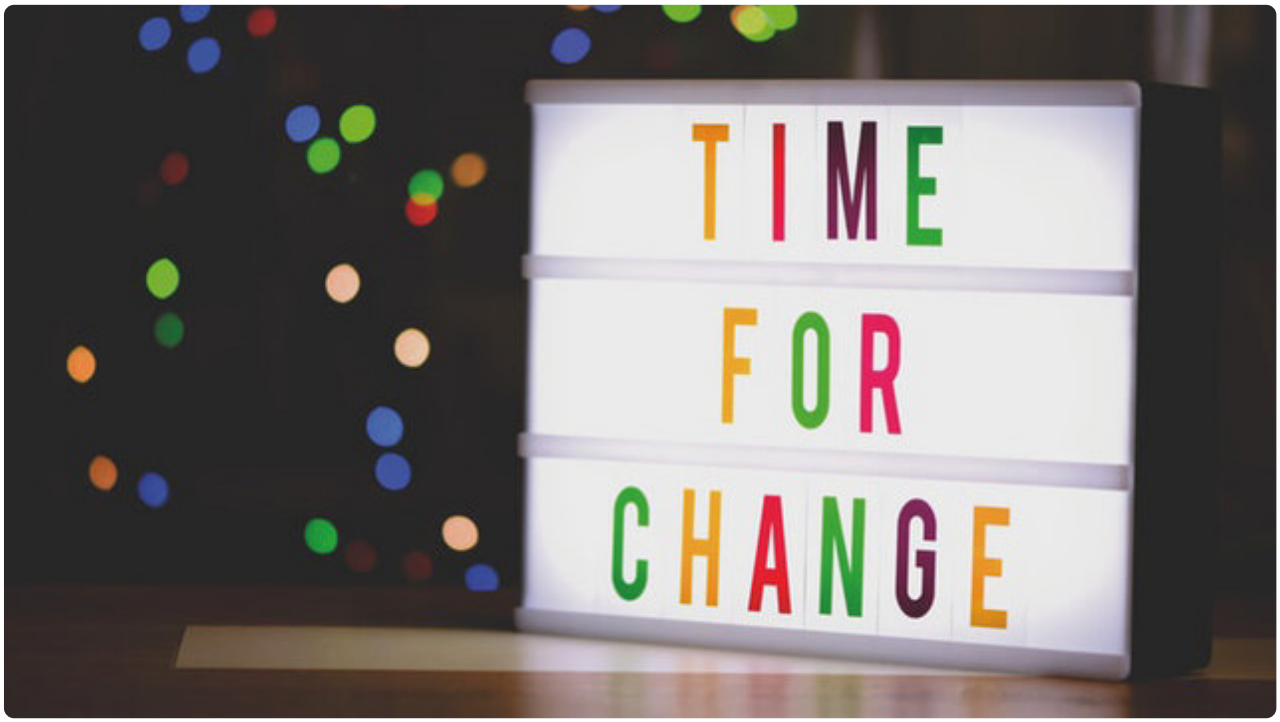
05 如何优化数据导入？

更新时间：2019-08-01 11:19:49



> 天才就是百分之二的灵感，百分之九十八的汗水。
>
> ——爱迪生

我们有时会遇到批量数据导入的场景，而数据量稍微大点，会发现导入非常耗时间。这篇文稿将介绍一些常用的加快数据导入的方法。

1 一次插入多行的值

插入行所需的时间由以下因素决定（参考MySQL 5.7参考手册：8.2.4.1优化INSERT语句）

- 连接：30%
- 向服务器发送查询：20%
- 解析查询：20%
- 插入行：10% * 行的大小
- 插入索引：10% * 索引数
- 结束：10%

可发现大部分时间耗费在客户端与服务端通信的时间，因此可以使用 insert 包含多个值来减少客户端和服务器之间的通信。我们通过实验来验证下一次插入多行与一次插入一行的效率对比。

1.1 准备测试表及数据

由于本次实验操作包含 drop 等危险操作，因此建议创建普通用户进行实验，并权限最小化，以防误操作。

```
grant select,insert,create,drop,index,alter on muke.* to 'test_user3'@'127.0.0.1' identified by 'userBcdQ19lc';
```

这里对上面命令解释一下：

参数	解释
grant	这里包含了创建用户及赋权操作
select,insert,create,drop	赋予查询、写入、建表及删表的权限
on muke.*	只对muke这个database有这些权限
to ‘test_user3’@‘127.0.0.1’	用户名为test_user3，host为127.0.0.1，即test_user3只能在本机使用，如果MySQL服务端跟客户端不在同一台机器上，则127.0.0.1替换成客户端ip地址
identified by ‘userBcdQ19lc’	用户密码为userBcdQ19lc

创建测试表及写入数据，语句如下

```
use muke;               /* 使用muke这个database */

drop table if exists t1;  /* 如果表t1存在则删除表t1 */

CREATE TABLE `t1` (       /* 创建表t1 */
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `a` varchar(20) DEFAULT NULL,
  `b` int(20) DEFAULT NULL,
  `c` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB CHARSET=utf8mb4 ;

drop procedure if exists insert_t1; /* 如果存在存储过程insert_t1，则删除 */
delimiter ;;
create procedure insert_t1()       /* 创建存储过程insert_t1 */
begin
  declare i int;                /* 声明变量i */
  set i=1;                      /* 设置i的初始值为1 */
  while(i<=10000)do         /* 对满足i<=1000的值进行while循环 */
    insert into t1(a,b) values(i,i); /* 写入表t1中a、b两个字段，值都为i当前的值 */
    set i=i+1;                  /* 将i加1 */
  end while;
end;;
delimiter ;
call insert_t1();             /* 运行存储过程insert_t1 */
```

1.2 导出一条 SQL 包含多行数据的数据文件

为了获取批量导入数据的 SQL，首先对测试表的数据进行备份，备份的 SQL 为一条 SQL 包含多行数据的形式（执行环境为 Centos7 命令行）。

```
[root@mysqltest muke]# mysqldump -utest_user3 -p'userBcdQ19lc' -h127.0.0.1 --set-gtid-purged=off --single-transaction --skip-add-locks  muke t1 >t1.sql
```

这里对上面 mysqldump 所使用到的一些参数做下解释：

参数	详解
-utest_user3	用户名，这里使用的是root用户
-p’userBcdQ19lc’	密码
-h127.0.0.1	连接的MySQL服务端IP
set-gtid-purged=off	不添加SET @@GLOBAL.GTID_PURGED
–single-transaction	设置事务的隔离级别为可重复读，即REPEATABLE READ，这样能保证在一个事务中所有相同的查询读取到同样的数据
–skip-add-locks	取消每个表导出之前加lock tables操作。
muke	库名
t1	表名

参数	详解
t1.sql	导出数据到这个文件

查看文件 t1.sql 内容，可看到数据是单条 SQL 有多条数据，如下：

```
.....
DROP TABLE IF EXISTS `t1`;
/* 按照上面的备份语句备份的数据文件包含drop命令时，需要特别小心，在后续使用备份文件做导入操作时，应该确定所有表名，防止drop掉业务正在使用的表 */
.....
CREATE TABLE `t1`.....
.....
INSERT INTO `t1` VALUES (1,'1',1,'2019-05-24 15:44:10'),(2,'2',2,'2019-05-24 15:44:10'),(3,'3',3,'2019-05-24 15:44:10').....
.....
```

1.3 导出一条SQL只包含一行数据的数据文件

```
[root@mysqltest muke]# mysqldump -utest_user3 -p'userBcdQ19lc' -h127.0.0.1 --set-gtid-purged=off --single-transaction --skip-add-locks --skip-extended-insert muke t1 >t1_row.sql
```

mysqldump命令参数解释：

参数	详解
–skip-extended-insert	一条SQL一行数据的形式导出数据

备份文件t1_row.sql内容如下

```
.....
INSERT INTO `t1` VALUES (1,'1',1,'2019-05-24 15:44:10');
INSERT INTO `t1` VALUES (2,'2',2,'2019-05-24 15:44:10');
INSERT INTO `t1` VALUES (3,'3',3,'2019-05-24 15:44:10');
.....
```

1.4 导入时间的对比

首先导入一行 SQL 包含多行数据的数据文件：

```
[root@mysqltest ~]# time mysql -utest_user3 -p'userBcdQ19lc' -h127.0.0.1 muke <t1.sql

real 0m0.230s
user 0m0.007s
sys  0m0.003s
```

耗时0.2秒左右。

导入一条SQL只包含一行数据的数据文件：

```
[root@mysqltest ~]# time mysql -utest_user3 -p'userBcdQ19lc' -h127.0.0.1 muke <t1_row.sql

real 0m31.138s
user 0m0.088s
sys  0m0.126s
```

耗时31秒左右。

1.5 结论

一次插入多行花费时间0.2秒，一次插入一行花费了31秒，对比效果明显，因此建议**有大批量导入时，推荐一条insert语句插入多行数据。**

2 关闭自动提交

2.1 对比开启和关闭自动提交的效率

Autocommit 开启时会为每个插入执行提交。可以在InnoDB导入数据时，关闭自动提交。如下：

```
SET autocommit=0;
INSERT INTO `t1` VALUES (1,'1',1,'2019-05-24 15:44:10');
INSERT INTO `t1` VALUES (2,'2',2,'2019-05-24 15:44:10');
INSERT INTO `t1` VALUES (3,'3',3,'2019-05-24 15:44:10');
......
COMMIT;
```

使用1.3 生成的t1_row.sql，在insert前增加：

```
SET autocommit=0;
```

在insert语句后面增加：

```
COMMIT;
```

```
[root@mysqltest muke]# time mysql -utest_user3 -p'userBcdQ19lc' -h127.0.0.1 muke <t1_row.sql

real    0m1.036s
user    0m0.062s
sys     0m0.108s
```

开启自动提交的情况下导入是31秒（执行详情见1.4 导入时间的对比）。

关闭自动提交的情况下导入是1秒左右，因此导入多条数据时，关闭自动提交，让多条 insert 一次提交，可以大大提升导入速度。

2.2 原因分析

与本节前面讲的一次插入多行能提高批量插入速度的原因一样，因为批量导入大部分时间耗费在客户端与服务端通信的时间，所以多条 insert 语句合并提交可以减少客户端与服务端通信的时间，并且合并提交还可以减少数据落盘的次数。

3 参数调整

影响MySQL写入速度的主要两个参数：innodb_flush_log_at_trx_commit、sync_binlog。

3.1 参数解释

innodb_flush_log_at_trx_commit：控制重做日志刷新到磁盘的策略，有0 、1和2三种值。

- 0：master线程每秒把redo log buffer写到操作系统缓存，再刷到磁盘；
- 1：每次提交事务都将redo log buffer写到操作系统缓存，再刷到磁盘；
- 2：每次事务提交都将redo log buffer写到操作系统缓存，由操作系统来管理刷盘。

备注：具体原理会在后续的事务这章进行详细描述。

sync_binlog：控制binlog的刷盘时机，可配置0、1或者大于1的数字。

- 0：二进制日志从不同步到磁盘，依赖OS刷盘机制；
- 1：二进制日志每次提交都会刷盘；
- n(n>1)：每n次提交落盘一次。

3.2 写入速度测试

为了对比以上两个参数对MySQL写入速度的影响，我们通过压力工具sysbench测试写入速度。

sysbench安装及使用请参考：http://imysql.cn/node/312。

第一步：设置两个参数的值：

```
mysql> set global innodb_flush_log_at_trx_commit=1;
Query OK, 0 rows affected (0.01 sec)

mysql> set global sync_binlog=1;
Query OK, 0 rows affected (0.00 sec)
```

第二步：准备数据：

```
[root@mysqltest ~]# sysbench --test=/usr/share/sysbench/tests/include/oltp_legacy/insert.lua --mysql-user=test_user3 --mysql-password='userBcdQ1
9lc' --mysql-host=127.0.0.1 --mysql-db=muke --oltp-table-size=0 --oltp-tables-count=10  prepare
```

第三步：进行写入测试：

```
[root@mysqltest ~]# sysbench --test=/usr/share/sysbench/tests/include/oltp_legacy/insert.lua --mysql-user=test_user3 --mysql-password='userBcdQ1
9lc' --mysql-host=127.0.0.1  --mysql-db=muke --oltp-table-size=100000 --max-time=100 --oltp-tables-count=10  run
```

> 正式环境压测建议压半小时以上。

第四步：记录测试结果：

```
[root@mysqltest ~]# sysbench --test=/usr/share/sysbench/tests/include/oltp_legacy/insert.lua --mysql-user=test_user3 --mysql-password='user
BcdQ19Ic' --mysql-host=127.0.0.1  --mysql-db=muke --oltp-table-size=100000 --max-time=100 --oltp-tables-count=10  run
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.
WARNING: --max-time is deprecated, use --time instead
sysbench 1.0.17 (using system LuaJIT 2.0.4)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Initializing worker threads...

Threads started!

SQL statistics:
    queries performed:
        read:                            0
        write:                           31684
        other:                           0
        total:                           31684
    transactions:                        31684  (316.83 per sec.)
    queries:                             31684  (316.83 per sec.)
    ignored errors:                      0      (0.00 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          100.0017s
    total number of events:              31684

Latency (ms):
         min:                                    1.81
         avg:                                    3.15
         max:                                  383.62
         95th percentile:                        5.28
         sum:                                99914.25

Threads fairness:
    events (avg/stddev):           31684.0000/0.00
    execution time (avg/stddev):   99.9143/0.00
```

TPS

第五步：清除压测数据：

```
[root@mysqltest ~]# sysbench --test=/usr/share/sysbench/tests/include/oltp_legacy/insert.lua --mysql-user=test_user3 --mysql-password='userBcdQ1
9Ic' --mysql-host=127.0.0.1  --mysql-db=muke --oltp-table-size=100000 --oltp-tables-count=10  cleanup
```

测试结果如下：

innodb_flush_log_at_trx_commit	sync_binlog	TPS	结论
1	1	316.83	双一情况写入速度最慢
1	0	526.97	
0	1	497.42	
0	0	2379.9	都设置为0的情况下，写入速度最快
2	1	515.76	
2	0	2169.51	

3.3 结论

从实验结果可以看出，innodb_flush_log_at_trx_commit设置为0、同时sync_binlog设置为0时，写入数据的速度是最快的。**如果对数据库安全性要求不高(比如你的测试环境)，可以尝试都设置为0后再导入数据，能大大提升导入速度**。

4 总结

今天一起研究了怎样提高 MySQL 批量导入数据的速度。根据测试，总结了加快批量数据导入有如下方法：

- 一次插入多行的值；
- 关闭自动提交，多次插入数据的 SQL 一次提交；
- 调整参数，innodb_flush_log_at_trx_commit 和 sync_binlog 都设置为0（当然这种情况可能会丢数据）。

5 问题

你一般工作中使用什么方法加快大批量数据导入的速度？

6 参考资料

唐汉明 等著；深入浅出MySQL（第2版）：18.4.2 优化INSERT语句

MySQL 5.7参考手册：8.2.4.1优化INSERT语句

sysbench的安装和做性能测试：http://imysql.cn/node/312

}

04 条件字段有索引，为什么查询也这么慢?

06 让order by、group by查询更快