# 前言

使用tpc-c的标准对mycat进行测试，以发现mycat对事务支持的bug和mycat事务使用中的注意事项，同时，通过让mycat通过tpcc这种标准企业级测试，让使用mycat的人群更有信心来使用mycat 。

# 测试方法概述

## 测试环境

## 硬件环境

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **机器ip** | **OS** | **磁盘大小** | **cpu** | **内存** | **网络** |
| 172.17.209.104 | Red Hat Enterprise Linux Server release 6.3 | 1.6T | Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz | 32G |  |
| 172.17.209.103 | Red Hat Enterprise Linux Server release 6.3 | 1.6T | Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz | 32G |  |
| 172.17.209.102 | Red Hat Enterprise Linux Server release 6.3 | 1.6T | Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz | 32G |  |
| 172.17.209.111 | Red Hat Enterprise Linux Server release 6.3 | 1.6T | Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz | 32G |  |
| 172.17.209.69  (虚拟机) | Red Hat Enterprise Linux Server release 6.3 | 500G | 8核 | 8G |  |

## 软件环境

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **软件名称** | **版本** | **用途** | **备注** |
| 1 | Jdk | 1.7.67 |  |  |
| 2 | Tpcc | 5.11 |  |  |
| 3 | Ganglia | 3.6.0 |  |  |
| 4 | Mycat | 1.3.02(2015-01-14) |  |  |
| 5 | Mysql | 5.6.19 |  |  |

## 测试工具

### TPCC介绍

TPC(Tracsaction Processing Performance Council) 事务处理性能协会是一个评价大型数据库系统软硬件性能的非盈利的组织,TPC-C是TPC协会制定的，用来测试典型的复杂OLTP系统的性能。

Tpcc测试结果主要关注两个指标：流量指标和性价比。

**流量指标**：（Throughput，简称tpmC）

系统每分钟能够执行多少个用户订单请求，即在执行支付（Payment）、订单状态查询（Order-status）、发货（Delivery）、库存状态查询（Stock-Level）这四种交易的同事，每分钟可以处理多少个新订单（New-Order）交易。

每分钟能处理的新订单事务数越大，说明数据库管理系统的性能越好，例如：

每分钟能处理的新订单事务数:13.6 个。

**性价比(Price/Performance，**简称**Price/tpmC)** ：

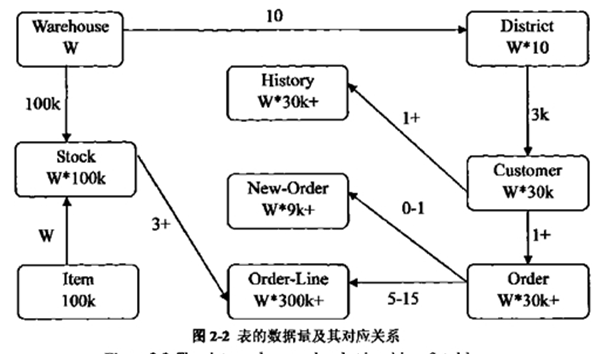
系统价格（指在美国的报价）与流量指标的比值

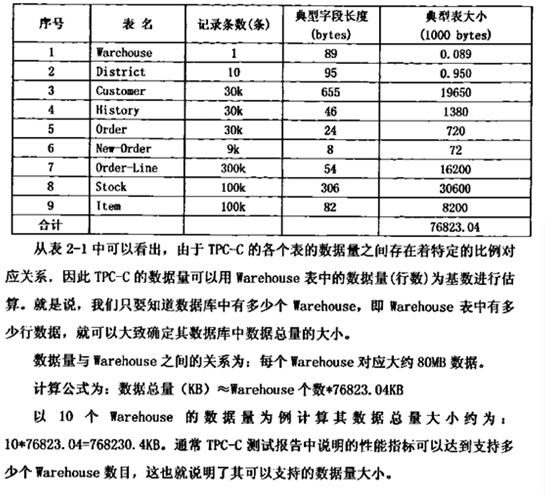
#### Tpcc商业模型

测试用到的模型是一个大型的批发销售公司，每个仓库负责10个区域的供货，每个区域为3000个客户提供服务。每个仓库维护公司销售的100,000种商品的库存记录，随着企业并发用户的增加，数据量也逐渐增加。商业模型示意图如下：



#### Tpcc表关系及数据量关系





#### 模拟事务

* **新订单**

从固定的仓库中随机选取5-15件商品，创建新订单。

* **支付操作**

采用随机的金额支付一笔订单，并作相应历史记录。

* **订单状态查询**

显示订单内每件商品的状态。

* **发货**

随机选取一个发货包，更新被处理订单的用户帐户余额

* **库存状态查询**

检查订单中所有货物的库存。

#### Tpcc客户端工具

使用开源项目：<https://github.com/codefutures/tpcc>

本次测试mycat使用的是fork后的修改版，地址为<https://github.com/wdw1206/tpcc>。

**对原项目修改的内容**：

（1）、有些地方加了注解，如stmt.execute("/\*!mycat: sql = select count(\*) from orders for update \*/SET UNIQUE\_CHECKS=0")

（2）、增加几个工具类：CountTable、TruncateTable、DropTable。

因为使用脚本文件没法将表删除干净，外键约束和事务导致。

（3）、将原来的数据库连接方式从DriverManager.getConnection(jdbcUrl, jdbcConnectProp)改成druid连接池中获取连接。因为tpcc事务运行时间很长时，由于原生连接不会忘服务端发心跳语句，导致超时被服务端自动断开连接。

**Java TPC-C**

This project is a Java implementation of the TPC-C benchmark.

**个性化修改内容：**使用该项目并做了个性化修改，主要是加了数据库连接池（防止连接操作断开连接），修复了里面的main函数启动（shell脚本启动main函数时的args）参数检验提示错误，以及针对mycat对一些语句加了注解。

#### TPCC测试结果如何分析？

**Java tpcc客户端输出结果形式如下：**

<Raw Results>

|NewOrder| sc:9826 lt:0 rt:0 fl:0

|Payment| sc:9813 lt:0 rt:0 fl:0

|Order Stat| sc:982 lt:0 rt:0 fl:0

|Delivery| sc:981 lt:0 rt:0 fl:0

|Slev| sc:982 lt:0 rt:0 fl:0

in 120.889999 sec.

<Raw Results2(sum ver.)>

|NewOrder| sc:9826 lt:0 rt:0 fl:0

|Payment| sc:9814 lt:0 rt:0 fl:0

|Order Stat| sc:982 lt:0 rt:0 fl:0

|Delivery| sc:982 lt:0 rt:0 fl:0

|Slev| sc:982 lt:0 rt:0 fl:0

<Constraint Check> (**all must be [OK]**)

[transaction percentage]

Payment: 43.449192% (>=43.0%) [OK]

Order-Status: 4.348019% (>= 4.0%) [OK]

Delivery: 4.343591% (>= 4.0%) [OK]

Stock-Level: 4.348019% (>= 4.0%) [OK]

[response time (at least 90% passed)]

NewOrder: 100.000000% [OK]

Payment: 100.000000% [OK]

Order Stat: 100.000000% [OK]

Delivery: 100.000000% [OK]

Slev: 100.000000% [OK]

NewOrder Total: 9831

Payment Total: 9813

Order Stat Total: 982

Delivery Total: 981

Slev Total: 982

<TpmC>

4879.3115 TpmC

**含义解释：**

New-Order       :新订单

Payment         :支付

Order-Status    :订单查询

Delivery        :发货

Stock-Level     :库存

|NewOrder| sc:9826 lt:0 rt:0 fl:0

分别表示success，late，retry，failure（成功、延迟、重试、失败）

RAMP-UP TIME.(120 sec.)  --热身时间

**总结**：

Tpcc测试结果包含两个方面：

1. **事务通过**。All master be OK.

Java tpcc程序直接报错中断，显然就是不通过，需要把错误解决了再测试。

Java tpcc程序不中断，且最终显示了上面的报告，就看看Constraint Check是否都是OK的。如果都是OK，就是通过，否则不通过。

不通过的情况，java tpcc程序的后台肯定有什么报错信息，如果只是因为服务器压力太大导致的延迟或者重试，需要调整压力再测试，或者增强服务器的处理能力再试。

<Constraint Check> (**all must be [OK]**)

[transaction percentage]

Payment: 43.449192% (>=43.0%) [OK]

Order-Status: 4.348019% (>= 4.0%) [OK]

Delivery: 4.343591% (>= 4.0%) [OK]

Stock-Level: 4.348019% (>= 4.0%) [OK]

[response time (at least 90% passed)]

NewOrder: 100.000000% [OK]

Payment: 100.000000% [OK]

Order Stat: 100.000000% [OK]

Delivery: 100.000000% [OK]

Slev: 100.000000% [OK]

1. **TpmC值越高越好**。

主要看TpmC的数值。这个在前文有介绍，见前文的“**流量指标”。**该指标越大越好，表示每分钟能处理新订单数。

### Ganglia介绍

主要是用来监控服务器性能，如：cpu 、mem、硬盘利用率， I/O负载、网络流量情况等。

本次没采集磁盘信息，由于主要测试事务的支持，没有极限压力测试。

## 测试方法

参照标准TPC-C方案，针对10仓库、20仓库、30仓库、60仓库、90，160仓库这几个不同级别的数据量进行测试。

# 测试过程

## 测试脚本准备



Drop table、Truncate table等操作由于涉及事务和外键约束，用脚本无法执行成功，改成java程序执行。见[3.5测试步骤](#_测试步骤)。

## Tpcc客户端程序验证

目的：验证tpcc客户端程序连接mysql是否能跑通。

在使用tpcc测试mycat之前需要，使用tpcc测试mysql验证tpcc程序是否正确，经验证，tpcc客户端程序测试mysql正常。

### 验证结果

通过

## Tpcc客户端程序+mycat验证

目的：验证tpcc客户端程序连接mycat是否能跑通。

### 使用分片策略

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表名 | 是否分片 | 拆分字段 | 拆分规则 | 拆分算法 | 主键 |
| customer | 是 | c\_w\_id | sharding\_by\_c\_w\_id | mod | c\_id |
| district | 是 | d\_w\_id | sharding\_by\_d\_w\_id | mod |  |
| history | 是 |  | sharding\_by\_h\_w\_id | mod |  |
| item | 是 |  |  | global |  |
| new\_orders | 是 | no\_w\_id | sharding\_by\_no\_w\_id | mod |  |
| order\_line | 是 | ol\_w\_id | sharding\_by\_ol\_w\_id | mod | ex\_id |
| orders | 是 | o\_w\_id | sharding\_by\_o\_w\_id | mod |  |
| stock | 是 | s\_w\_id | sharding\_by\_s\_w\_id | mod | h\_t\_id |
| warehouse | 是 | w\_id | sharding\_by\_w\_id | mod | w\_id |

所有表都分片：数据量少的使用全局表，数据量大的使用仓库id取模分片。

### 验证结果

通过

## 测试工具开发

### 测试脚本



该脚本执行前需要配置环境变量：

vi /etc/profile

增加以下内容：

export CLASSPATH=$CLASSPATH:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar:/data1/tpcctest/tpcc-1.0.0-SNAPSHOT-jar-with-dependencies.jar

tpcc-1.0.0-SNAPSHOT-jar-with-dependencies.jar为java tpcc打包后的jar文件（打包使用命令：

mvn package assembly:assembly，必须使用这样的命令，因为要把依赖的jar全部打进去）

### 配置mycat配置文件

## 测试步骤

步骤：

1. 手工创建database，如有3个分片，需要把3个分片的database全部创建好，否则mycat启动就会报错。
2. 配置好schema.xml、rule、server.xml，启动mycat。
3. 创建表。使用脚本create\_tables.sql。

命令：execute\_sql < create\_tables.sql

1. 创建索引、外键

命令：execute\_sql < add\_fkey\_idx.sql

1. LoadData。使用命令：

java com.codefutures.tpcc.TpccLoad -l "$tar\_jdbc\_url" -u ${tar\_db\_user} -p ${tar\_db\_pass} -m JDBC -w $num\_wh -s 1 -i 1

1. 运行主测试程序。使用命令：

java com.codefutures.tpcc.Tpcc -l "$tar\_jdbc\_url" -u ${tar\_db\_user} -p ${tar\_db\_pass} -w $num\_wh -c $num\_con -r $ramp\_up -t $duration

1. 收集监控数据（cpu\_report、mem\_report），填写测试报告。
2. 统计运行完成后的所有表的记录数。

命令：java com.codefutures.tpcc.util.CountTable -l "$tar\_jdbc\_url" -u ${tar\_db\_user} -p ${tar\_db\_pass}

1. Drop table。

命令：

java com.codefutures.tpcc.util.DropTable -l "$tar\_jdbc\_url" -u ${tar\_db\_user} -p ${tar\_db\_pass}

命令细节见[3.4.1测试脚本](#_测试脚本)

# 测试结果

## 数量级10仓库3分片（800M）

|  |  |  |  |
| --- | --- | --- | --- |
| 10仓库3分片，事务运行30分钟 | | | |
| 表名 | 记录条数 | 最大单表记录数 | 占用空间 |
| warehouse | 10 | 1000000 | 800M |
| district | 100 |
| customer | 300000 |
| history | 300000 |
| order | 300000 |
| new-order | 90000 |
| order-line | 3000000 |
| stock | 1000000 |
| item | 100000 |

### Tpcc结果

#### Load数据

...DATA LOADING COMPLETED SUCCESSFULLY.

Total execution time: 7 minute(s), 26 second(s) (7.433 minutes)

real 7m27.513s

user 2m18.231s

sys 0m5.540s

#### 运行事务输出结果

---------------------------------------------------

<Raw Results>

|NewOrder| sc:758579 lt:16 rt:0 fl:0

|Payment| sc:758575 lt:13 rt:0 fl:0

|Order Stat| sc:75862 lt:0 rt:0 fl:0

|Delivery| sc:75860 lt:1 rt:0 fl:0

|Slev| sc:75862 lt:0 rt:0 fl:0

in 1800.859009 sec.

<Raw Results2(sum ver.)>

|NewOrder| sc:758593 lt:16 rt:0 fl:0

|Payment| sc:758590 lt:13 rt:0 fl:0

|Order Stat| sc:75862 lt:0 rt:0 fl:0

|Delivery| sc:75860 lt:1 rt:0 fl:0

|Slev| sc:75862 lt:0 rt:0 fl:0

<Constraint Check> (all must be [OK])

[transaction percentage]

Payment: 43.477880% (>=43.0%) [OK]

Order-Status: 4.347966% (>= 4.0%) [OK]

Delivery: 4.347908% (>= 4.0%) [OK]

Stock-Level: 4.347966% (>= 4.0%) [OK]

[response time (at least 90% passed)]

NewOrder: 99.997891% [OK]

Payment: 99.998286% [OK]

Order Stat: 100.000000% [OK]

Delivery: 99.998682% [OK]

Slev: 100.000000% [OK]

NewOrder Total: 758597

Payment Total: 758590

Order Stat Total: 75862

Delivery Total: 75861

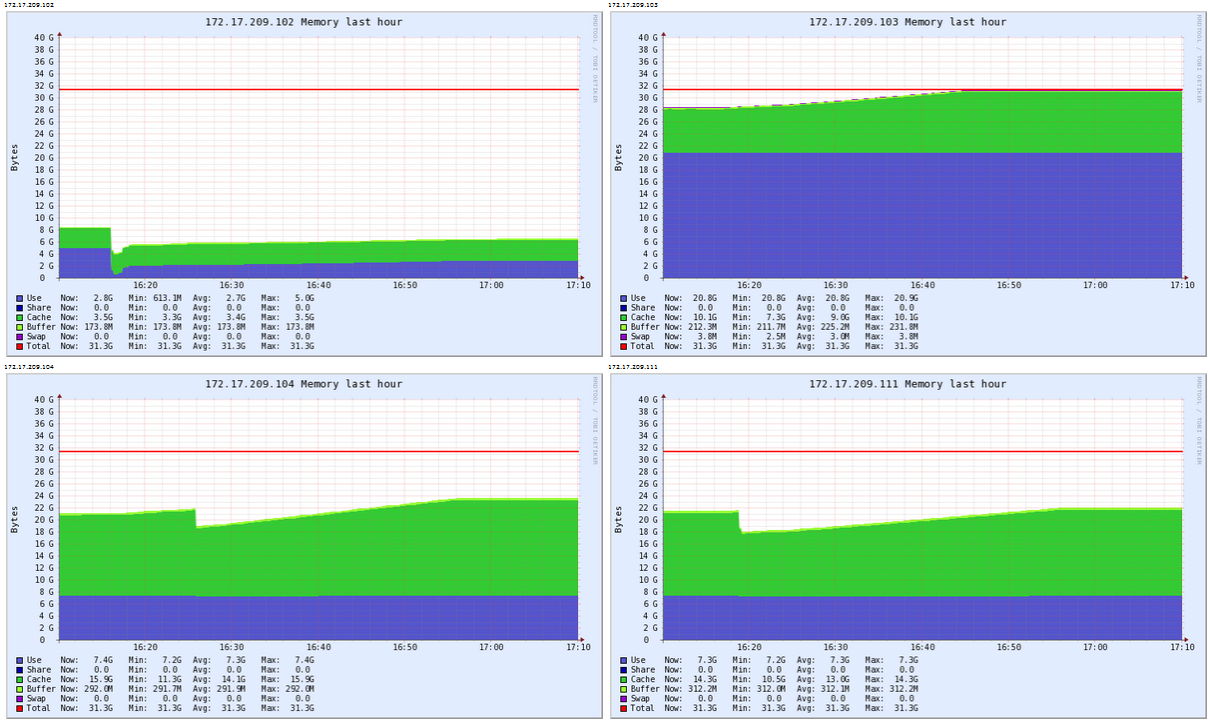
Slev Total: 75862

<TpmC>

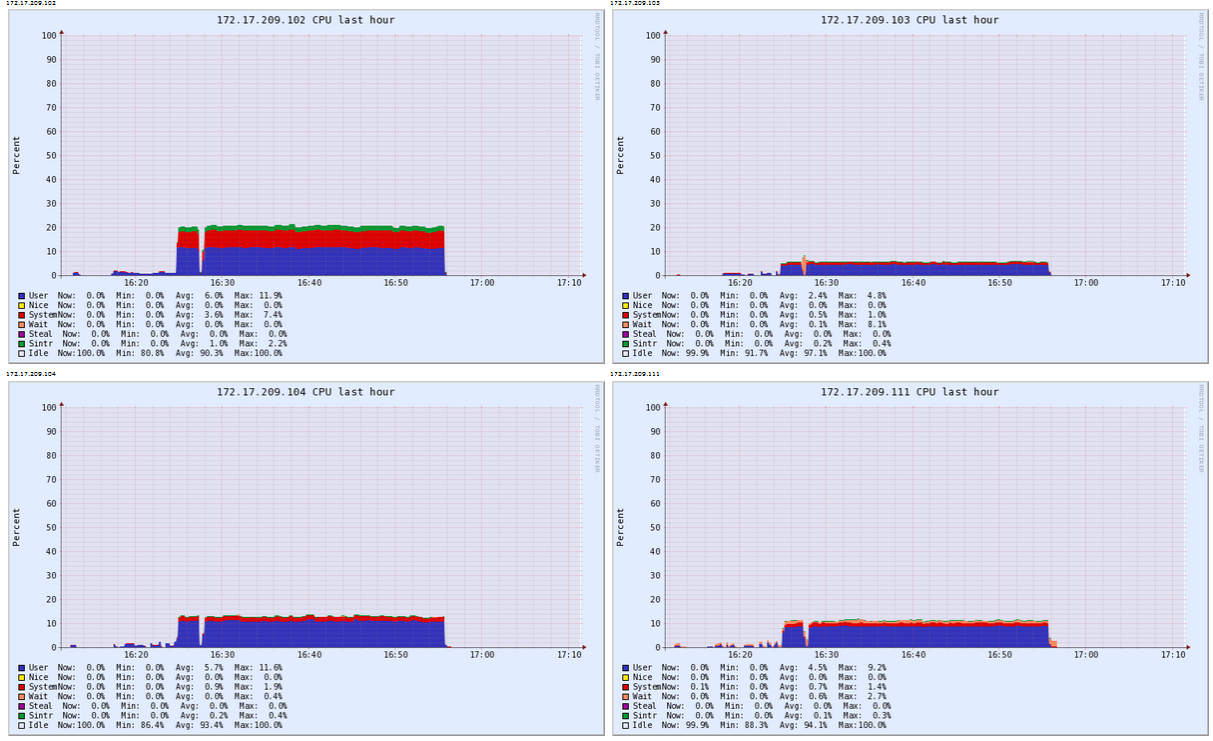
**25274.506 TpmC**

### 服务器资源消耗

#### 内存



#### Cpu



## 数据量级20仓库3分片（1.6G）

|  |  |  |  |
| --- | --- | --- | --- |
| 20仓库3分片，事务运行50分钟 | | | |
| 表名 | 记录条数 | 最大单表记录数 | 占用空间 |
| warehouse | 20 | 2000000 | 1.6G |
| district | 200 |
| customer | 600000 |
| history | 600000 |
| order | 600000 |
| new-order | 180000 |
| order-line | 6000000 |
| stock | 2000000 |
| item | 100000 |

### Tpcc测试结果

#### Load数据

...DATA LOADING COMPLETED SUCCESSFULLY.

Total execution time: 14 minute(s), 42 second(s) (14.700 minutes)

real 14m43.641s

user 4m22.802s

sys 0m10.028s

#### 运行事务输出结果

---------------------------------------------------

<Raw Results>

|NewOrder| sc:1414492 lt:0 rt:0 fl:0

|Payment| sc:1414525 lt:0 rt:0 fl:0

|Order Stat| sc:141455 lt:0 rt:0 fl:0

|Delivery| sc:141454 lt:0 rt:0 fl:0

|Slev| sc:141455 lt:0 rt:0 fl:0

in 3000.357910 sec.

<Raw Results2(sum ver.)>

|NewOrder| sc:1414548 lt:0 rt:0 fl:0

|Payment| sc:1414548 lt:0 rt:0 fl:0

|Order Stat| sc:141455 lt:0 rt:0 fl:0

|Delivery| sc:141455 lt:0 rt:0 fl:0

|Slev| sc:141455 lt:0 rt:0 fl:0

<Constraint Check> (all must be [OK])

[transaction percentage]

Payment: 43.478610% (>=43.0%) [OK]

Order-Status: 4.347932% (>= 4.0%) [OK]

Delivery: 4.347901% (>= 4.0%) [OK]

Stock-Level: 4.347932% (>= 4.0%) [OK]

[response time (at least 90% passed)]

NewOrder: 100.000000% [OK]

Payment: 100.000000% [OK]

Order Stat: 100.000000% [OK]

Delivery: 100.000000% [OK]

Slev: 100.000000% [OK]

NewOrder Total: 1414498

Payment Total: 1414532

Order Stat Total: 141456

Delivery Total: 141455

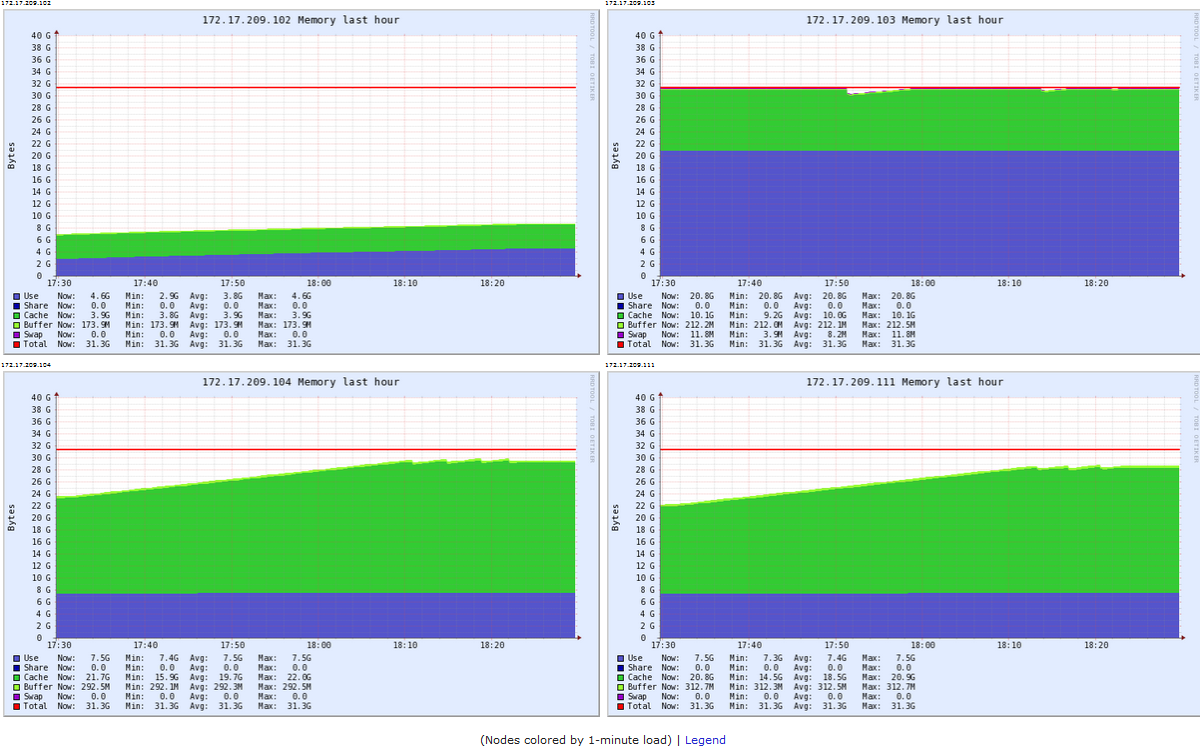
Slev Total: 141455

<TpmC>

28286.586 TpmC

### 服务器资源消耗

#### 内存



#### Cpu



## 数据量级30仓库3分片（2.4G）

|  |  |  |  |
| --- | --- | --- | --- |
| 30仓库3分片,运行2小时 | | | |
| 表名 | 记录条数 | 最大单表记录数 | 占用空间 |
| warehouse | 30 | 3000000 | 2.4G |
| district | 300 |
| customer | 900000 |
| history | 900000 |
| order | 900000 |
| new-order | 270000 |
| order-line | 9000000 |
| stock | 3000000 |
| item | 100000 |

### Tpcc测试结果

#### Load数据

...DATA LOADING COMPLETED SUCCESSFULLY.

Total execution time: 21 minute(s), 52 second(s) (21.867 minutes)

real 21m53.151s

user 6m33.593s

sys 0m14.685s

#### 运行事务输出结果

---------------------------------------------------

<Raw Results>

|NewOrder| sc:3488016 lt:0 rt:0 fl:0

|Payment| sc:3488127 lt:0 rt:0 fl:0

|Order Stat| sc:348819 lt:0 rt:0 fl:0

|Delivery| sc:348815 lt:0 rt:0 fl:0

|Slev| sc:348817 lt:0 rt:0 fl:0

in 7200.091797 sec.

<Raw Results2(sum ver.)>

|NewOrder| sc:3488189 lt:0 rt:0 fl:0

|Payment| sc:3488183 lt:0 rt:0 fl:0

|Order Stat| sc:348819 lt:0 rt:0 fl:0

|Delivery| sc:348818 lt:0 rt:0 fl:0

|Slev| sc:348818 lt:0 rt:0 fl:0

<Constraint Check> (all must be [OK])

[transaction percentage]

Payment: 43.478782% (>=43.0%) [OK]

Order-Status: 4.347957% (>= 4.0%) [OK]

Delivery: 4.347907% (>= 4.0%) [OK]

Stock-Level: 4.347932% (>= 4.0%) [OK]

[response time (at least 90% passed)]

NewOrder: 100.000000% [OK]

Payment: 100.000000% [OK]

Order Stat: 100.000000% [OK]

Delivery: 100.000000% [OK]

Slev: 100.000000% [OK]

NewOrder Total: 3488018

Payment Total: 3488127

Order Stat Total: 348819

Delivery Total: 348815

Slev Total: 348817

<TpmC>

29066.443 TpmC

---------------------------------------------------

<Raw Results>

|NewOrder| sc:3468211 lt:44 rt:0 fl:0

|Payment| sc:3468351 lt:10 rt:0 fl:0

|Order Stat| sc:346838 lt:2 rt:0 fl:0

|Delivery| sc:346836 lt:4 rt:0 fl:0

|Slev| sc:346842 lt:0 rt:0 fl:0

in 7200.250977 sec.

<Raw Results2(sum ver.)>

|NewOrder| sc:3468373 lt:44 rt:0 fl:0

|Payment| sc:3468394 lt:10 rt:0 fl:0

|Order Stat| sc:346839 lt:2 rt:0 fl:0

|Delivery| sc:346839 lt:4 rt:0 fl:0

|Slev| sc:346842 lt:0 rt:0 fl:0

<Constraint Check> (all must be [OK])

[transaction percentage]

Payment: 43.478764% (>=43.0%) [OK]

Order-Status: 4.347925% (>= 4.0%) [OK]

Delivery: 4.347925% (>= 4.0%) [OK]

Stock-Level: 4.347950% (>= 4.0%) [OK]

[response time (at least 90% passed)]

NewOrder: 99.998731% [OK]

Payment: 99.999712% [OK]

Order Stat: 99.999423% [OK]

Delivery: 99.998847% [OK]

Slev: 100.000000% [OK]

NewOrder Total: 3468255

Payment Total: 3468361

Order Stat Total: 346840

Delivery Total: 346840

Slev Total: 346842

<TpmC>

28901.117 TpmC

### 运行完成时的数据量

customer:900000

district:300

history:4417344

item:100000

new\_orders:308800

order\_line:43817722

orders:4381780

stock:3000000

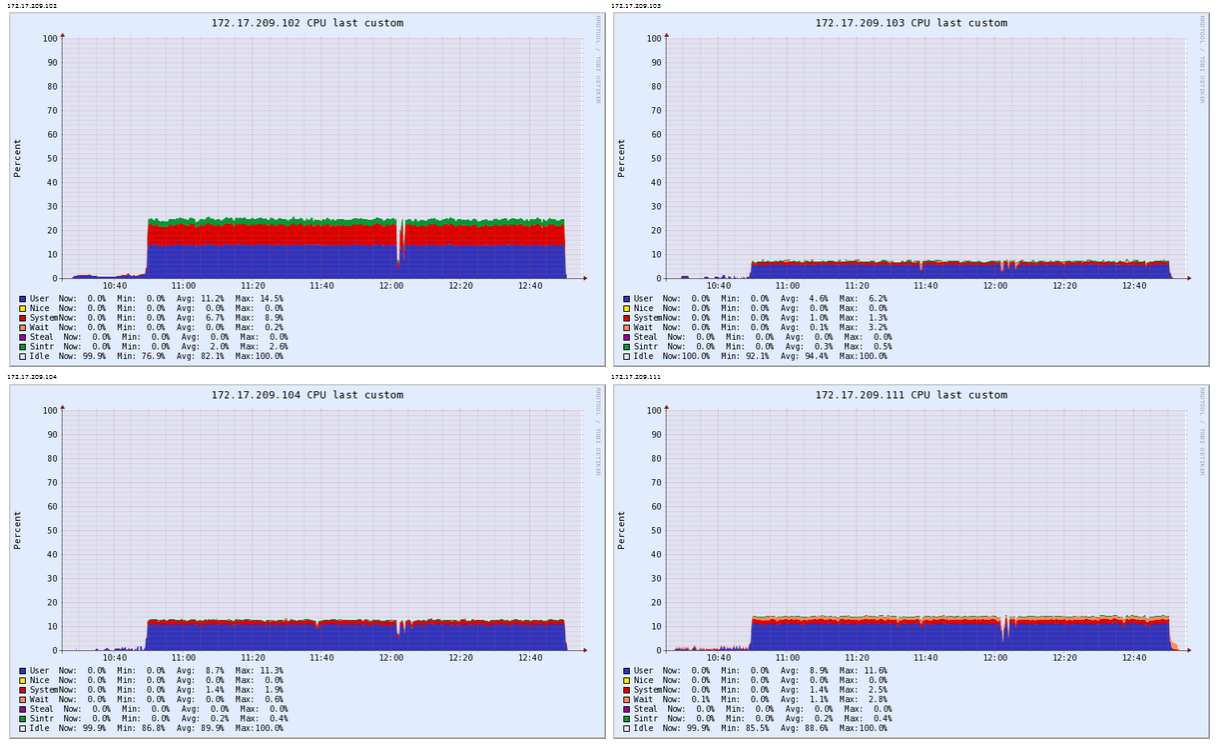
warehouse:30

### 服务器资源消耗

#### 内存



#### Cpu



## 数据量级50仓库3分片（4G）

|  |  |  |  |
| --- | --- | --- | --- |
| 50仓库3分片，运行1小时 | | | |
| 表名 | 记录条数 | 最大单表记录数 | 占用空间 |
| warehouse | 50 | 5000000 | 4G |
| district | 500 |
| customer | 1500000 |
| history | 1500000 |
| order | 1500000 |
| new-order | 450000 |
| order-line | 15000000 |
| stock | 5000000 |
| item | 100000 |

### Tpcc测试结果

#### Load数据

...DATA LOADING COMPLETED SUCCESSFULLY.

Total execution time: 36 minute(s), 30 second(s) (36.500 minutes)

real 36m31.831s

user 10m47.751s

sys 0m25.417s

#### 运行事务输出结果

---------------------------------------------------

<Raw Results>

|NewOrder| sc:2635750 lt:0 rt:0 fl:0

|Payment| sc:2635789 lt:0 rt:0 fl:0

|Order Stat| sc:263589 lt:0 rt:0 fl:0

|Delivery| sc:263588 lt:0 rt:0 fl:0

|Slev| sc:263587 lt:0 rt:0 fl:0

in 3600.712891 sec.

<Raw Results2(sum ver.)>

|NewOrder| sc:2635878 lt:0 rt:0 fl:0

|Payment| sc:2635883 lt:0 rt:0 fl:0

|Order Stat| sc:263589 lt:0 rt:0 fl:0

|Delivery| sc:263590 lt:0 rt:0 fl:0

|Slev| sc:263588 lt:0 rt:0 fl:0

<Constraint Check> (all must be [OK])

[transaction percentage]

Payment: 43.478320% (>=43.0%) [OK]

Order-Status: 4.348007% (>= 4.0%) [OK]

Delivery: 4.347974% (>= 4.0%) [OK]

Stock-Level: 4.347974% (>= 4.0%) [OK]

[response time (at least 90% passed)]

NewOrder: 100.000000% [OK]

Payment: 100.000000% [OK]

Order Stat: 100.000000% [OK]

Delivery: 100.000000% [OK]

Slev: 100.000000% [OK]

NewOrder Total: 2635767

Payment Total: 2635799

Order Stat Total: 263590

Delivery Total: 263588

Slev Total: 263589

<TpmC>

43920.77 TpmC

### 运行完成时数据量

customer:1500000

district:500

history:4230851

item:100000

new\_orders:487158

order\_line:42029179

orders:4203528

stock:5000000

warehouse:50

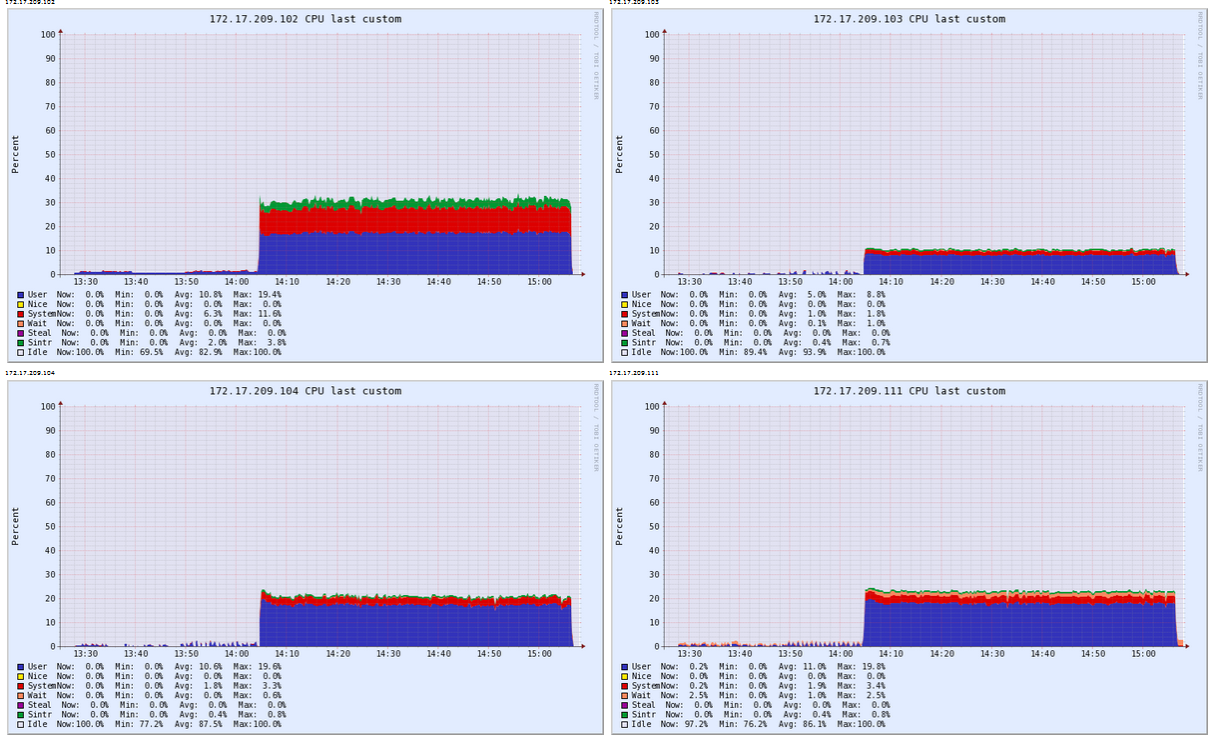
最大单表数据量：14293640

### 服务器资源消耗

#### 内存



#### Cpu



## 数据量级90仓库3分片（7.2G）

|  |  |  |  |
| --- | --- | --- | --- |
| 90仓库3分片,运行1小时 | | | |
| 表名 | 记录条数 | 最大单表记录数 | 占用空间 |
| warehouse | 90 | 9000000 | 7.2G |
| district | 900 |
| customer | 2700000 |
| history | 2700000 |
| order | 2700000 |
| new-order | 810000 |
| order-line | 27000000 |
| stock | 9000000 |
| item | 100000 |

### Tpcc测试结果

#### Load数据

...DATA LOADING COMPLETED SUCCESSFULLY.

Total execution time: 65 minute(s), 29 second(s) (65.483 minutes)

real 65m30.826s

user 19m19.772s

sys 0m44.829s

#### 运行事务输出结果

---------------------------------------------------

<Raw Results>

|NewOrder| sc:3037928 lt:74 rt:0 fl:0

|Payment| sc:3038034 lt:37 rt:0 fl:0

|Order Stat| sc:303821 lt:2 rt:0 fl:0

|Delivery| sc:303814 lt:7 rt:0 fl:0

|Slev| sc:303824 lt:0 rt:0 fl:0

in 3600.675049 sec.

<Raw Results2(sum ver.)>

|NewOrder| sc:3038167 lt:74 rt:0 fl:0

|Payment| sc:3038203 lt:37 rt:0 fl:0

|Order Stat| sc:303822 lt:2 rt:0 fl:0

|Delivery| sc:303817 lt:7 rt:0 fl:0

|Slev| sc:303824 lt:0 rt:0 fl:0

<Constraint Check> (all must be [OK])

[transaction percentage]

Payment: 43.478403% (>=43.0%) [OK]

Order-Status: 4.348075% (>= 4.0%) [OK]

Delivery: 4.348046% (>= 4.0%) [OK]

Stock-Level: 4.348075% (>= 4.0%) [OK]

[response time (at least 90% passed)]

NewOrder: 99.997564% [OK]

Payment: 99.998782% [OK]

Order Stat: 99.999342% [OK]

Delivery: 99.997696% [OK]

Slev: 100.000000% [OK]

NewOrder Total: 3038010

Payment Total: 3038080

Order Stat Total: 303824

Delivery Total: 303823

Slev Total: 303824

<TpmC>

50624.027 TpmC

### 运行完成时数据量

customer:2700000

district:900

history:5846018

item:100000

new\_orders:825502

order\_line:58155151

orders:5814463

stock:9000000

warehouse:90

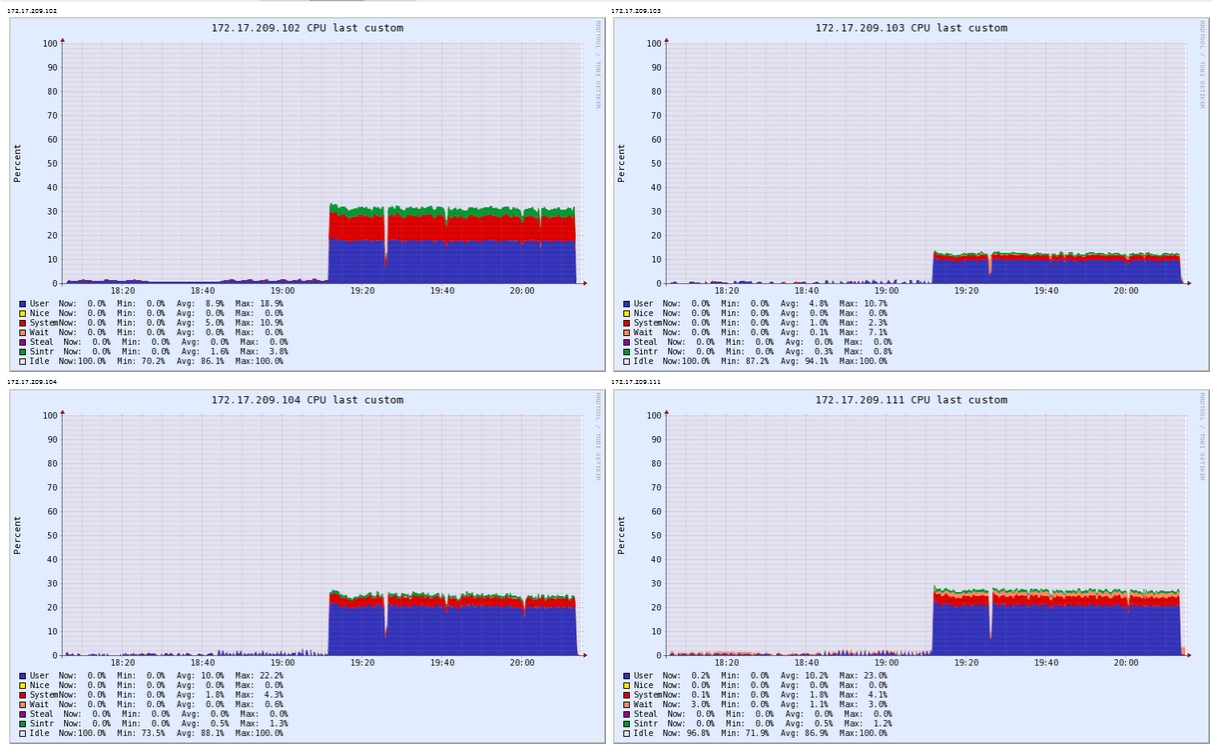
最大单表记录数：19383057

### 服务器资源消耗

#### 内存



#### Cpu



## 数据量级160仓库3分片（12.8G）

|  |  |  |  |
| --- | --- | --- | --- |
| 160仓库，3分片，100并发，运行3小时 | | | |
| 表名 | 记录条数 | 最大单表记录数 | 占用空间 |
| warehouse | 160 | 16000000 | 12.8G |
| district | 1600 |
| customer | 4800000 |
| history | 4800000 |
| order | 4800000 |
| new-order | 1440000 |
| order-line | 48000000 |
| stock | 16000000 |
| item | 100000 |

### Tpcc测试结果

#### Load数据

...DATA LOADING COMPLETED SUCCESSFULLY.

Total execution time: 118 minute(s), 14 second(s) (118.233 minutes)

real 118m16.010s

user 35m2.463s

sys 1m17.348s

#### 运行事务输出结果

---------------------------------------------------

<Raw Results>

|NewOrder| sc:11582090 lt:1690 rt:4 fl:0

|Payment| sc:11583460 lt:556 rt:11 fl:0

|Order Stat| sc:1158431 lt:28 rt:0 fl:0

|Delivery| sc:1158285 lt:163 rt:1 fl:0

|Slev| sc:1158460 lt:0 rt:0 fl:0

in 10800.097656 sec.

<Raw Results2(sum ver.)>

|NewOrder| sc:11582935 lt:1691 rt:4 fl:0

|Payment| sc:11584062 lt:559 rt:11 fl:0

|Order Stat| sc:1158435 lt:28 rt:0 fl:0

|Delivery| sc:1158301 lt:163 rt:1 fl:0

|Slev| sc:1158461 lt:0 rt:0 fl:0

<Constraint Check> (all must be [OK])

[transaction percentage]

Payment: 43.478378% (>=43.0%) [OK]

Order-Status: 4.348056% (>= 4.0%) [OK]

Delivery: 4.348014% (>= 4.0%) [OK]

Stock-Level: 4.348056% (>= 4.0%) [OK]

[response time (at least 90% passed)]

NewOrder: 99.985411% [OK]

Payment: 99.995200% [OK]

Order Stat: 99.997583% [OK]

Delivery: 99.985929% [OK]

Slev: 100.000000% [OK]

NewOrder Total: 11583791

Payment Total: 11584026

Order Stat Total: 1158460

Delivery Total: 1158449

Slev Total: 1158461

<TpmC>

64353.81 TpmC

### 运行完成时数据量

customer:4800000

district:1600

history:16678455

item:100000

new\_orders:1488531

order\_line:165602445

orders:16559431

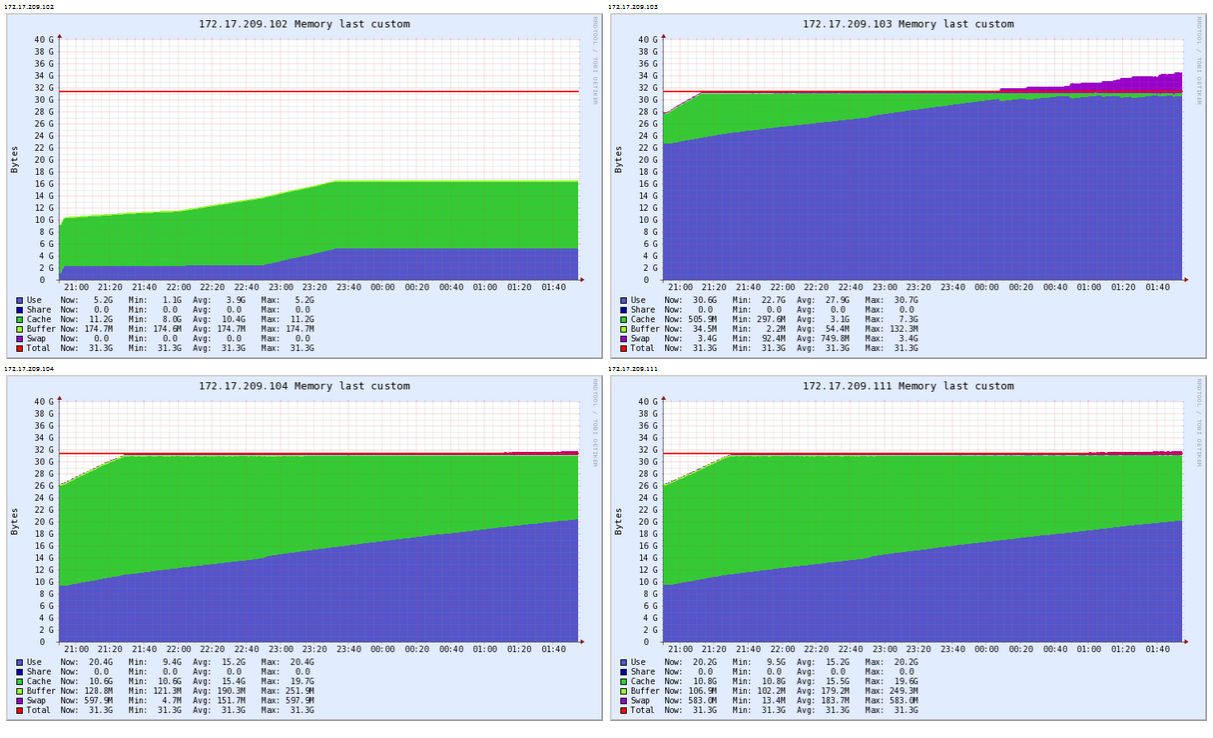
stock:16000000

warehouse:160

最大单表记录数：54833477

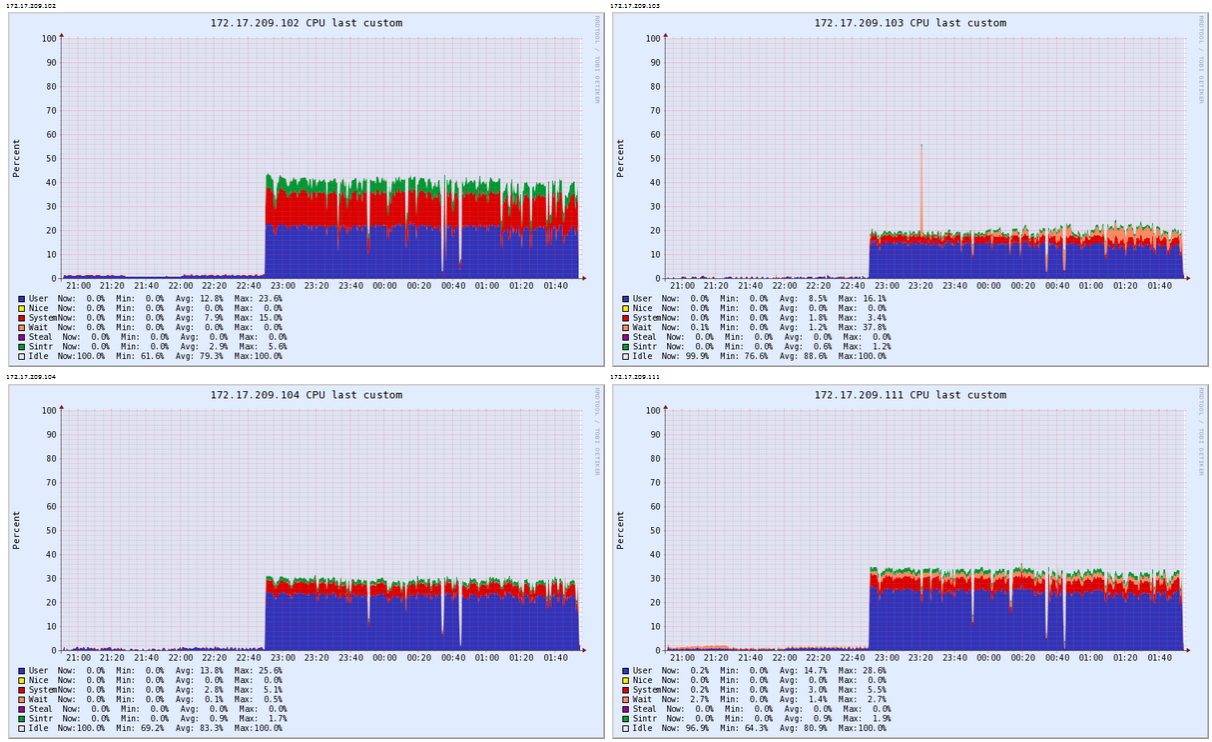
### 服务器资源消耗

#### 内存



后台有一台Mysql服务器（172.17.209.103）的内存快被用完。

#### Cpu



## 数据量级160仓库10分片（12.8G）

|  |  |  |  |
| --- | --- | --- | --- |
| 160仓库，10分片，100并发，运行3小时 | | | |
| 表名 | 记录条数 | 最大单表记录数 | 占用空间 |
| warehouse | 160 | 16000000 | 12.8G |
| district | 1600 |
| customer | 4800000 |
| history | 4800000 |
| order | 4800000 |
| new-order | 1440000 |
| order-line | 48000000 |
| stock | 16000000 |
| item | 100000 |

### Tpcc测试结果

#### Load数据

...DATA LOADING COMPLETED SUCCESSFULLY.

Total execution time: 107 minute(s), 27 second(s) (107.450 minutes)

real 107m28.583s

user 34m26.455s

sys 1m17.180s

#### 运行事务输出结果

---------------------------------------------------

<Raw Results>

|NewOrder| sc:9193496 lt:4721 rt:0 fl:0

|Payment| sc:9196429 lt:1972 rt:0 fl:0

|Order Stat| sc:919685 lt:183 rt:0 fl:0

|Delivery| sc:919288 lt:570 rt:0 fl:0

|Slev| sc:919876 lt:0 rt:0 fl:0

in 10800.322266 sec.

<Raw Results2(sum ver.)>

|NewOrder| sc:9194054 lt:4721 rt:0 fl:0

|Payment| sc:9196770 lt:1972 rt:0 fl:0

|Order Stat| sc:919686 lt:183 rt:0 fl:0

|Delivery| sc:919307 lt:570 rt:0 fl:0

|Slev| sc:919876 lt:0 rt:0 fl:0

<Constraint Check> (all must be [OK])

[transaction percentage]

Payment: 43.478471% (>=43.0%) [OK]

Order-Status: 4.347979% (>= 4.0%) [OK]

Delivery: 4.347932% (>= 4.0%) [OK]

Stock-Level: 4.348017% (>= 4.0%) [OK]

[response time (at least 90% passed)]

NewOrder: 99.948675% [OK]

Payment: 99.978561% [OK]

Order Stat: 99.980106% [OK]

Delivery: 99.938034% [OK]

Slev: 100.000000% [OK]

NewOrder Total: 9198217

Payment Total: 9198401

Order Stat Total: 919868

Delivery Total: 919858

Slev Total: 919876

<TpmC>

51099.68 TpmC

### 运行完成时数据量

customer:4800000

district:1600

history: 15218552

item:100000

new\_orders: 1567206

order\_line: 151155479

orders: 15114299

stock: 16000000

warehouse:160

最大单表记录数：15118091

### 服务器资源消耗

#### 内存



后台有一台Mysql服务器（172.17.209.103）的内存被用完,出现swap。

#### Cpu



# 测试总结

Mycat目前的版本（1.3.0.2）要跑通tpcc测试存在一些问题，但通过修改mycat源码或者客户端编程注意避免可以使之通过。

## 测试发现的有关mycat的事务bug

1. Select …for update语句无法加锁。
2. SET UNIQUE\_CHECKS=0这样的语句无法直接支持事务，需要做调整（后面具体讲解怎么调整支持）
3. 全局表多线程update死锁。

## Mycat编程式事务----陷阱，如何避免？

### Mycat编程式事务流程



该流程与其他数据库（如mysql）没什么区别，但mycat事务与普通mysql事务还是有些区别的，主要是因为mycat内部涉及前端连接（FrontendConnection）和后端连接（BackendConnection），而如果直连mysql，则只有一个连接，不区分什么前端后端的。前端连接指的是从客户端连接到mycat的连接，后端连接指的是从mycat连接到后台mysql的连接。Mycat的后端连接是一个连接池，每次执行语句，会从后端连接池中取一个连接与前端连接绑定，如果是开启事务，执行完语句后前后端连接的绑定不会解绑，即后端连接不会释放，如果不开启事务，执行完语句后端连接马上就是放掉。

### Mycat开启事务的条件

**Mycat开启事务有两个必要条件**：autocommit==false并且canRunInReadDB==true。

**开启事务的表现为**：前后端连接处于绑定状态，后端连接不释放，必须调用con.commit()才释放后端连接。

解释下canRunInReadDB==true这个条件的意思。字面上表示能在读库上执行，mycat的读库是readOnly的，写语句都不能在读库上执行。所以canRunInReadDB==true表示sql是写语句,也叫modifySQL，写语句包含哪些在此就不解释了。事务开启就是必须autocommit==false并且碰到写语句，只要执行过写语句，前后端连接就会一直绑定，知道执行con.commit()。

**Mycat的特殊之处：**如果一个事务内全部是普通select语句（只读语句），即使设置了autocommit==false，对于mycat来说事务都是相当于不开启的，因为无法满足canRunInReadDB==true的条件。

#### 如何控制事务开启？

Mycat可以控制sql的语句类型来改变是否开启事务。如都是普通select语句，我也可以让它开启事务。

**应用程序客户端控制事务开启**：

connection.setAutocommit(false)。

加注解改变sql语句类型。如

/\*!mycat: sql = update district set d\_name = 'lqz' WHERE d\_id = 1 AND d\_w\_id = 4 \*/SELECT d\_next\_o\_id FROM district WHERE d\_id = 1 AND d\_w\_id = 4 FOR UPDATE

将一个select…for update语句改成update语句类型，使之开启事务锁住行记录。

**Mycat服务端控制canRunInReadDB参数的值**：

通过路由解析器修改canRunInReadDB的值。如一个select语句，一般情况下canRunInReadDB的值为true，我们可以在解析时它修改为false。

这种方式的实用场景：select … for update语句需要开启事务，解析时将他的canRunInReadDB参数改成false即可开启事务。

### Mycat事务范围

Mycat事务范围在于写语句的路由范围，如果写语句路由到dn1、dn2，则连接到dn1、dn2的后端连接都会绑定，如果只路由到dn1，则连接到dn1的后端连接会绑定。

#### 如何控制mycat事务范围？

通过注解方式可以控制事务范围。如

/\*!mycat: sql = select count(\*) from item for update \*/SET UNIQUE\_CHECKS=0。

这个语句我们要使SET UNIQUE\_CHECKS=0语句在所有节点生效，就加上select count(\*) from item for update语句注解，这个语句范围是所有节点，而且select…for update是可以开启事务的语句类型。

# 测试碰到的问题及解决方案

## 客户端连接超时

Caused by: com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: The last packet successfully received from the server was 41,330,583 milliseconds ago. The last packet sent successfully to the server was 41,330,582 milliseconds ago. is longer than the server configured value of 'wait\_timeout'. You should consider either expiring and/or testing connection validity before use in your application, increasing the server configured values for client timeouts, or using the Connector/J connection property 'autoReconnect=true' to avoid this problem.

原因：客户端连接超时，连接被服务单主动断开。

解决办法：客户端定时向服务端发心跳语句。客户端使用连接池，连接池设置如下参数：

*dataSource*.setTestWhileIdle(**true**);

*dataSource*.setValidationQuery("select 1");

*dataSource*.setTimeBetweenEvictionRunsMillis(60000);

*dataSource*.setMinEvictableIdleTimeMillis(300000);

## Select for update 无法锁住行记录

SELECT d\_next\_o\_id FROM district WHERE d\_id = 1 AND d\_w\_id = 4 FOR UPDATE;

语句无法锁住行记录，导致

并发插入时报这个错： Caused by: com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException: Duplicate entry '4-10-3008' for k

ey 'PRIMARY'

at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)

at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:57)

at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)

at java.lang.reflect.C

pStmts[1] = prepareStatement("SELECT d\_next\_o\_id, d\_tax FROM district WHERE d\_id = ? AND d\_w\_id = ? FOR UPDATE");

pStmts[2] = prepareStatement("UPDATE district SET d\_next\_o\_id = ? + 1 WHERE d\_id = ? AND d\_w\_id = ?");

如订单编号都是从district表取号(d\_next\_o\_id的值)，然后再递增1，如果select …for update没有加锁，就会导致主键重复。

解决方法：修改mycat源码，或者加注解。

1. **修改mycat源码：**

解析到select …for update语句时，修改canRunInReadDB参数，这个参数能控制backendConnection不释放。

见org.opencloudb.parser.druid.impl. DruidSelectParser类中

**if**(mysqlSelectQuery.isForUpdate()) {

rrs.changeCanRunInReadDB(**false**);

}

1. **加注解：**

SELECT d\_next\_o\_id FROM district WHERE d\_id = 1 AND d\_w\_id = 4 FOR UPDATE;语句加注释（/\*!mycat: sql = UPDATE district SET d\_next\_o\_id = ? + 1 WHERE d\_id = ? AND d\_w\_id = ? \*/当然，这里的？必须替换成具体值，否则无法找到准确的分片）。

注解成update语句，update语句canRunInReadDB参数会自动设置成false， select结果集返回后，backendConnection就不会释放，就会使行记录处于锁定状态。

## 多线程update同全局表出现数据库死锁

前提是带事务的连接中（con.setAutoCommit(false);）执行update全局表，实际上这个场景可以总结为：一个路由到多分片执行的update语句，在多线程并发操作时会死锁。

**重现方式**：mycat中配置一个全局表，至少2个节点（越多越容易重现），启动50个线程（线程越多越容易重现），如果不重现，可以加大全局表的节点数，或者增加线程数。

**测试代码**： 

### 死锁原因分析

#### 死锁场景

为简化场景易于使大家理解，全局表节点数和线程数都用最少的数量。设全局表item（商品）只有两个节点dn1，dn2，

Schema.xml中配置为：

<table name="item" dataNode="dn1,dn2 " type = "global" />

#### 死锁流程

Java代码步骤：

conn = ConnectionManager.getConnection();

conn.setAutoCommit(false);

statement = conn.createStatement();

statement.execute("update item set i\_data='test' where i\_id=1");

conn.commit();



流程：

1. Thread1发送update语到dn1
2. Dn1返回update响应信息（此时dn1没有执行commit，所以其他的update语句发过来都会阻塞）
3. Thread2发送update语句到dn1
4. Thread2等待dn1的update响应信息---因为被Thread1发的update语句锁住了，只能等待Thread1发送commit命令释放锁
5. Thread2发送update语句到dn2（刚好比Thread1发的快了一点点）
6. Dn1返回update响应信息（此时dn1没有执行commit，所以其他的update语句发过来都会阻塞）
7. Thread1发送update语句到dn2
8. Thread1等待dn2的update响应信息---因为被Thread2发的update语句锁住了，只能等待Thread2发送commit命令释放锁
9. Thread1必须受到所有分片返回的update响应信息后才能发送commit命令，同样Thread2也是要等待所有分片返回update响应信息才能发送commit。
10. 这样就出现了Thread1等Thread2发送commit来释放dn2的锁，Thread2等Thread1发送commit来释放dn1的锁，互相等待，出现死锁。

### 这种死锁如何解决？----mycat修复or使用者避免

#### Mycat能否修复？代价？

Mycat修复个人感觉代价太大，也很难修复。一方面是性能下降很多，另一方面锁粒度不好控制，过来一个路由到多分片的update语句，mycat不知道具体数据，就不知道是加行级锁还是锁全表，锁全表显然性能就差的离谱了。

但是可以从客户端加锁的方式解决。

#### 使用者避免---客户端加锁

比如对上面的场景，如果是单服务器中的多线程程序，直接加synchronized锁或者Lock锁就能控制住，如果是分布式的，可以借助数据库锁或者分布式锁。数据库锁指的是利用另一张单节点表，这个表专门用来加锁用的，如表名为lock\_table，上面的java程序可以改成如下即可。

conn = ConnectionManager.getConnection();

conn.setAutoCommit(false);

statement = conn.createStatement();

//专门加锁,这一句就锁住了lock\_table中的一行记录。其他任何线程或者另一台服务器上的//线程再想锁住该行记录都会阻塞

statement.execute("update lock\_table set is\_locked=1 where table\_name=’item’");

statement.execute("update item set i\_data='test' where i\_id=1");

conn.commit();

## Java Tpcc客户端换成druid连接池后出现内存溢出

Caused by: java.lang.OutOfMemoryError: GC overhead limit exceeded

at com.mysql.jdbc.MysqlIO.readPacket(MysqlIO.java:596)

at com.mysql.jdbc.MysqlIO.getResultSet(MysqlIO.java:414)

at com.mysql.jdbc.MysqlIO.readResultsForQueryOrUpdate(MysqlIO.java:2997)

at com.mysql.jdbc.MysqlIO.readAllResults(MysqlIO.java:2245)

at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:2638)

at com.mysql.jdbc.ConnectionImpl.execSQL(ConnectionImpl.java:2530)

at com.mysql.jdbc.PreparedStatement.executeInternal(PreparedStatement.java:1907)

at com.mysql.jdbc.PreparedStatement.executeQuery(PreparedStatement.java:2030)

at com.alibaba.druid.pool.DruidPooledPreparedStatement.executeQuery(DruidPooledPreparedStatement.java:227)

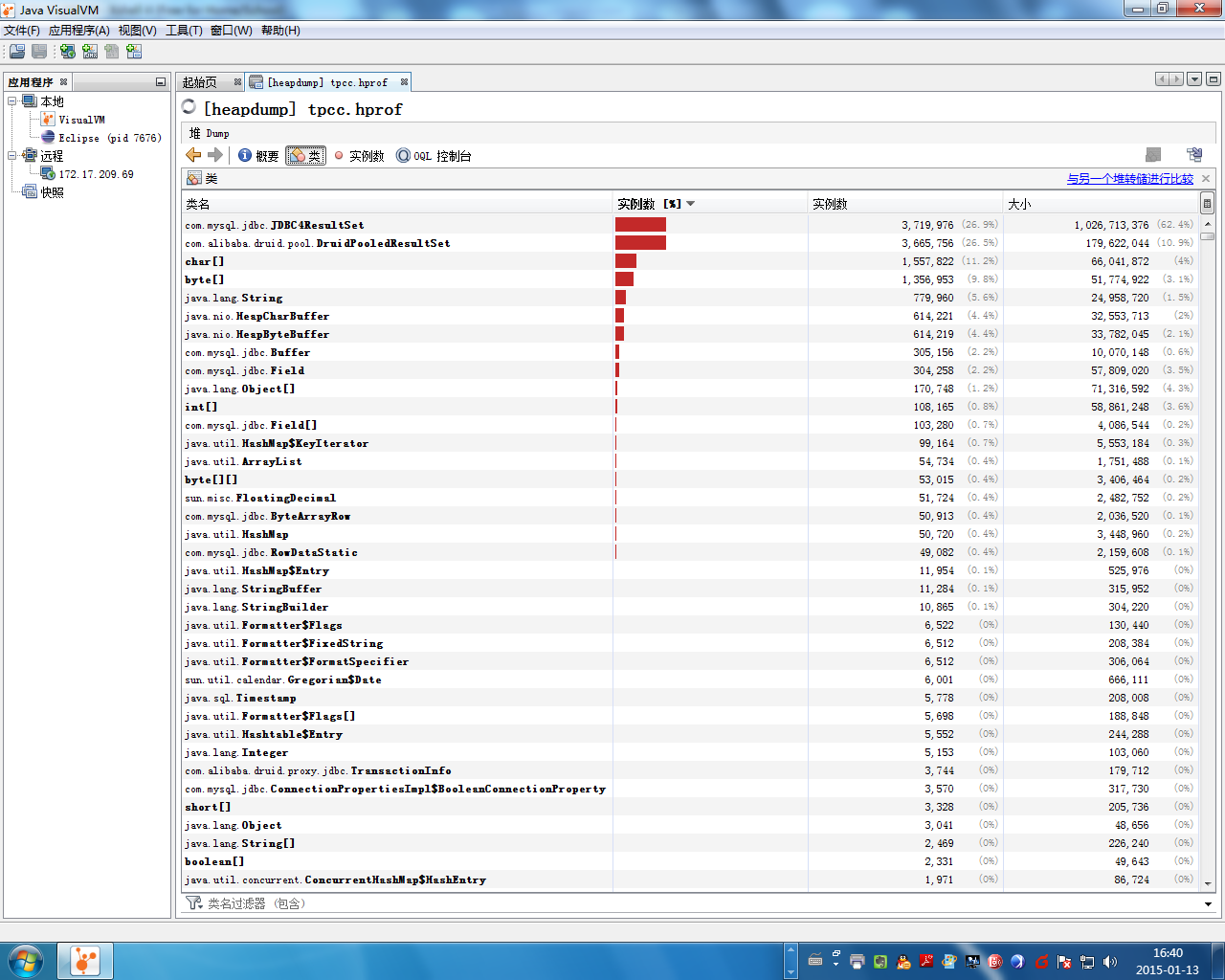
at com.codefutures.tpcc.Payment.payment(Payment.java:150)

**解决（步骤）方法：**

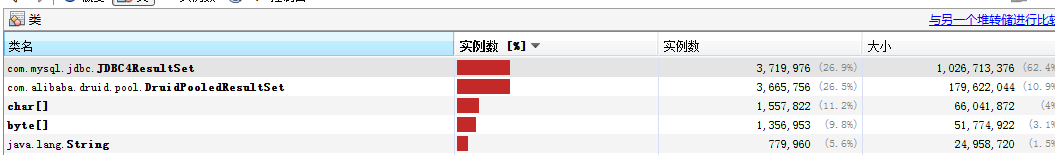
运行tpcc程序的过程中，使用

jmap -dump:format=b,file=dumpfile.hprof   [pid]命令导出内存镜像，

然后使用jvisualvm.exe载入分析。



如上图所示，实例最多的对象为



显然与ResultSet有关，经过查找代码，找到原因为com.alibaba.druid.pool. DruidPooledStatement类中protected List<ResultSet> resultSetTrace;属性中堆积了大量的ResultSet，因为每次executeQuery（sql）都会把ResultSet添加到resultSetTrace中，而tpcc程序中DruidPooledStatement的实例都是一直不close的，不close就不会清除resultSetTrace中的内容，所以导致了内存溢出。

**解决方式**：换其他连接池（换完还可能带来其他问题，所以就不换了，解决本次问题就行了），或者修改DruidPooledStatement类，注释掉resultSetTrace.add(resultSet);

**protected** **void** addResultSetTrace(ResultSet resultSet) {

**if** (resultSetTrace == **null**) {

resultSetTrace = **new** ArrayList<ResultSet>(1);

}

// resultSetTrace.add(resultSet);

}

## Load数据外键约束导致失败

Stock Done.

Loading District

Error loading data

java.lang.RuntimeException: Error loading into table 'district' with SQL: INSERT IGNORE INTO `district` (d\_id,d\_w\_id,d\_name,d\_street\_1,d\_street\_2,d\_city,d\_state,d\_zip,d\_tax,d\_ytd,d\_next\_o\_id) VALUES (1,2,'ub2lxdhjP','G0lro0pdFbMZDih','NrGKhHfwCiBWVx','7SIogSdUgp74NUC8Z2l2','8y','d64HNDrNn',0.15,30000.0,3001),(2,2,'GrUBI4VGD','6hzArXoKk6tkCicOeI3N','iDyeO9NlJWRBqtwcS','oL04SouPFp','Hr','bbMWJENkV',0.17,30000.0,3001),(3,2,'3iWnOQ0tiE','wWdhY1pxswApLjfA','mqnmGe3qcq','FxZ4o104XK','pu','t8F5aQfRU',0.2,30000.0,3001),(4,2,'VBTt2J','8YX5fqE2Xszr','2XJAYZSkUcPGH8MrS7Tw','3al1uBUqZ3e','Eq','PAOdVQp9E',0.2,30000.0,3001),(5,2,'7vESt7UN4','ZvRUnqFhTn','ZEqsWxgLs20H','KQF5xSKNWQlA','vV','87dnDSZ5w',0.17,30000.0,3001) ,(10,2,'aqklcynlc','lsU7itGwADozHUmL','ZXExQLqs8H2S1Y3L','5WIt4xfvDO','yA','D8ETy7qIL',0.15,30000.0,3001)

at com.codefutures.tpcc.load.JdbcStatementLoader.executeBulkInsert(JdbcStatementLoader.java:72)

at com.codefutures.tpcc.load.JdbcStatementLoader.close(JdbcStatementLoader.java:105)

at com.codefutures.tpcc.Load.district(Load.java:424)

at com.codefutures.tpcc.Load.loadWare(Load.java:180)

at com.codefutures.tpcc.TpccLoad.runLoad(TpccLoad.java:285)

at com.codefutures.tpcc.TpccLoad.main(TpccLoad.java:360)

Caused by: com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException: Cannot add or update a child row: a foreign key constraint fails (`tpcc\_db3`.`district`, CONSTRAINT `fkey\_district\_1` FOREIGN KEY (`d\_w\_id`) REFERENCES `warehouse` (`w\_id`))

at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)

at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:57)

at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)

at java.lang.reflect.Constructor.newInstance(Constructor.java:526)

at com.mysql.jdbc.Util.handleNewInstance(Util.java:377)

at com.mysql.jdbc.Util.getInstance(Util.java:360)

at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:971)

at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3887)

at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3823)

at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:2435)

at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:2582)

at com.mysql.jdbc.ConnectionImpl.execSQL(ConnectionImpl.java:2526)

at com.mysql.jdbc.ConnectionImpl.execSQL(ConnectionImpl.java:2484)

at com.mysql.jdbc.StatementImpl.execute(StatementImpl.java:848)

at com.mysql.jdbc.StatementImpl.execute(StatementImpl.java:742)

at com.alibaba.druid.pool.DruidPooledStatement.execute(DruidPooledStatement.java:421)

at com.codefutures.tpcc.load.JdbcStatementLoader.executeBulkInsert(JdbcStatementLoader.java:70)

... 5 more

该问题后来不重现了。

# TODO

待测试、待验证的问题。

1. TpccLoad执行时，有一定概率SET FOREIGN\_KEY\_CHECKS=0失效，同一个连接，同一个事务中应该是不会失效的。----该问题后来不重现了。
2. 目前mycat是不支持begin end等语法的，可以考虑修改mycat源码支持（该支持修改可能需要较多时间调试和测试，暂不考虑）。