dbproxy 的测试报告

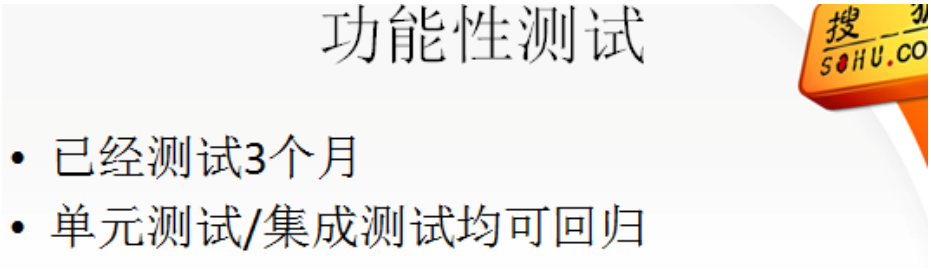
变更说明

日期	版本	变更位置	变更说明	作者
2013-9-12	0.1		初始文档	赖亿
2013-9-30	0.2			谢文，侯金轩，赖亿

目录

1 测试总结

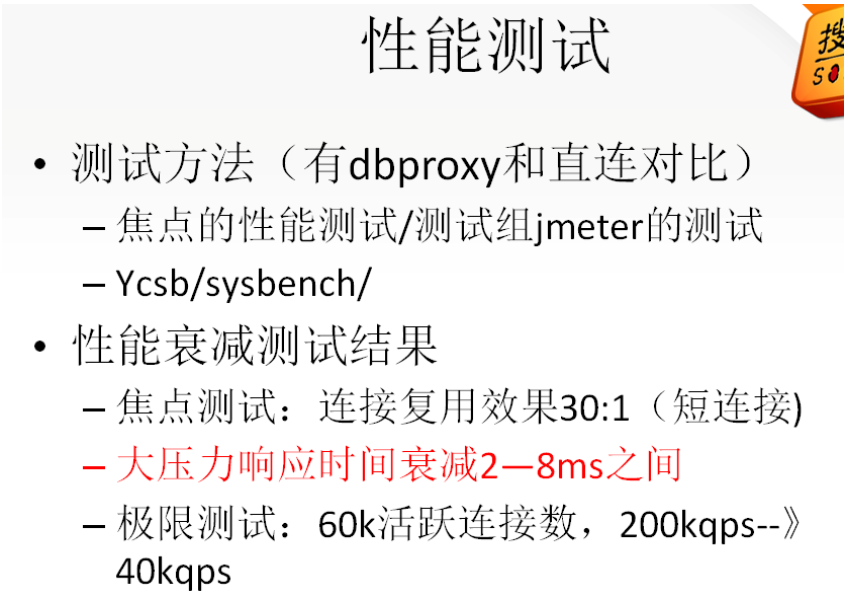
1.1 测试总结



1 功能性测试
- 1.1 单元测试 **通过 （共计 48 个单元测试用例）**
- 1.2 集成测试
  - 1.2.1 模块功能测试 **通过 （8 个测试套件，含 57 个测试用例）**
  - 1.2.2 MySQL 兼容性测试 **通过 （310 个测试用例）**
  - 1.2.3 有压力的情况下，结果集正确性测试 **通过**
- 1.3 焦点业务线的功能性测试 **通过**

2 稳定性测试 **通过（运行稳定，系统内存资源的占用没有明显增加）**

3 健壮性测试
- 3.1 网络异常模拟测试 **通过**



1.2 测试结果

1 功能性测试
- 1.1 单元测试 **通过 （共计 48 个单元测试用例）**
- 1.2 集成测试
  - 1.2.1 模块功能测试 **通过 （8 个测试套件，含 57 个测试用例）**
  - 1.2.2 MySQL 兼容性测试 **通过 （310 个测试用例）**

1.2.3 有压力的情况下，结果集正确性测试 **通过**

1.3 焦点业务线的功能性测试 **通过**

2 稳定性测试 **通过（运行稳定，系统内存资源的占用没有明显增加）**

3 健壮性测试

3.1 网络异常模拟测试 **通过**

4 性能测试

4.1 焦点的性能衰减测试 **通过（连接复用效果较好 30:1，后端观察到的 QPS80K，衰减 30%-50%）**

4.2 ycsb 性能衰减测试 **通过（QPS13K，性能衰减 10%以下。UPDATE 响应时间 100ms，和直连的差别不大。SELECT 的响应时间约 1 毫秒，相比直连的增加一倍）**

4.3 测试组 jmeter 的性能衰减测试 **通过（较高并发（2k 到 20k）情况下，proxy 的 QPS 比直连数据库性能好大约 25%，后端观察到的 QPS 保持在 20k 左右。mysql12k 个连接后 QPS 就有较大下降，proxy 超过 20k 个连接后 QPS 有较大下降）**

4.4 性能基准测试 **通过（随前端连接数增加 QPS 基本上呈线性增长，24 个连接数左右达到最大值 40K+QPS。然后前端一直加到 30K 连接数，QPS 保持平稳（后端 MySQL 开启了线程池管理）。但对比直连，QPS 有较大损耗（尤其是较高并发下），约 50%到 80%。响应时间也有 1 到 5 倍的增加，4K 连接数后超过 100 毫秒，30K 时接近 1 秒）**

2 功能性测试

2.1 单元测试

2.1.1 测试目的

单元测试由一组独立的测试构成，每个测试针对软件中的一个单独的程序单元。单元测试并非检查程序单元之间是否能够合作良好，而是检查单个程序单元行为是否正确。

首先保证代码质量。编写的代码虽然可以通过编译器检测到语法的正确性质，但并不能保证代码逻辑也是正确的。

其次保证代码的可维护。只要单元测试是正确的，那就可以保证无论怎么修改那段代码，只要测试代码通过测试，那么修改的逻辑代码是正确的。还有，比如，保证修改了这段代码不会影响到其他的模块等。

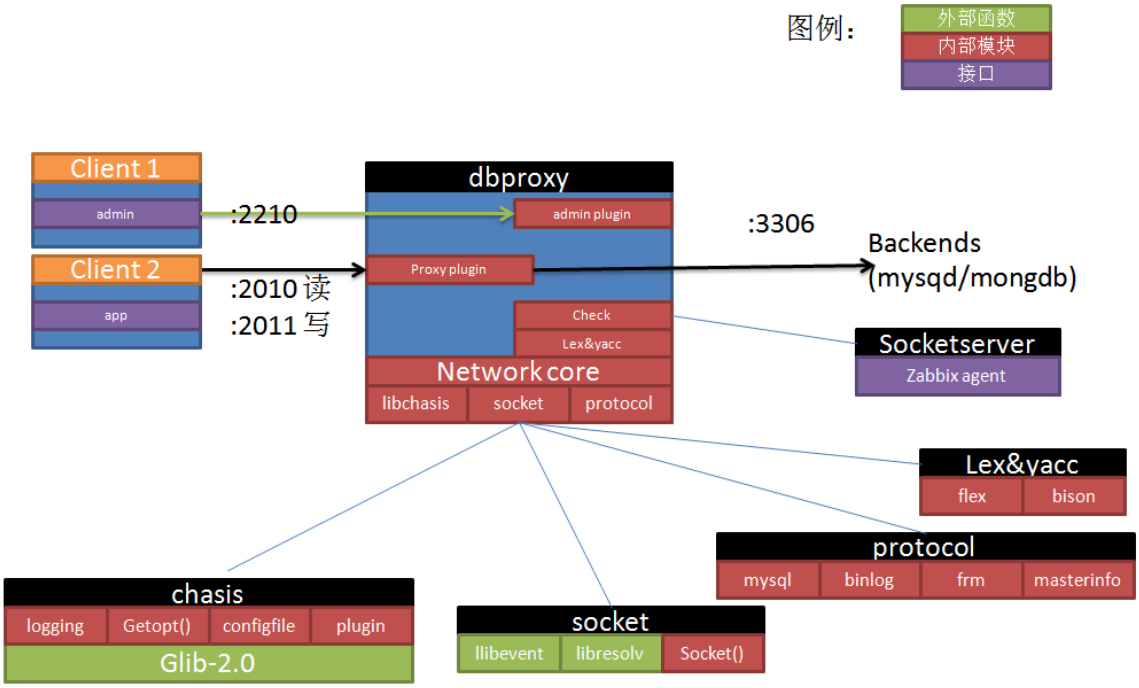
再次保证代码的可扩展。遵循一个代码之间的耦合度降到最低。单元测试对这方面有很强的好处，为了程序的可维护性，可以强迫开发人员编写低耦合度的程序。

2.1.2 测试结论和分析

对新增加和修改过的重要的程序模块编写了新的测试用例，另外包括 MySQL-Proxy 原有的测试用例，共计 48 个（列表如下），均测试通过。

测试用例包括了框架（chassis）和网络协议（network）、SQL 语句解析、admin 管理命令、外部脚本执行（zabbix）、健康检查和负载均衡等一些基础模块。

DBProxy 整体系统架构图：



2.1.3 测试环境

单元测试用例不依赖外部资源，在开发环境和持续集成服务器环境即可执行。
开发环境是 Oracle Linux 5.7，IP 地址 X.X.X.X
持续集成服务器环境 Oracle Linux 6.3，IP 地址 X.X.X.X/X.X.X.X

2.1.4 测试方法

单元测试用例由开发人员依据单元测试规范编写。当每次提交代码前在自己的开发环境下手工执行（make check），必须通过后才能提交到主干；每日构建最新 nightly 版本前也要自动执行，通过测试后编译形成最新的 nightly 版本上传到文件分发服务器。

具体参考 WIKI：开发规范，测试规范

2.2 集成测试

2.2.1 模块功能测试

2.2.1.1 测试目的

检查新增功能模块执行的正确性，是否能达到预期的结果。对单元测试的补充，一些新增模块之间存在相互调用关系，或者依赖外部资源，只能靠集成测试保证其正确性。

检验之前发生过的 BUG 的修改是否正确。是否能达到预期目的，不影响其它功能模块的正确性。

2.2.1.2 测试结论和分析

对新增加的主要功能模块都编写了相应了集成测试脚本，测试的功能模块包括：
user_conn_limit：前端连接限制测试模块
conn_multiplex：连接复用测试模块
conn_context：上下文恢复测试模块
bk_async_con：后端连接异步建立模块
rw_ro_service：读写端口测试模块
load_balance：负载均衡测试模块
bug_test：bug 列表测试模块
admin_mange：管理端口测试模块

测试通过率 100%说明主要功能模块达成了预期的目标，也确认了之前出现的一些 BUG 没有再现。

2.2.1.3 测试环境

测试脚本对环境有所依赖，在开发环境下执行测试通过。
后端数据库 IP 地址：X.X.X.X 和 X.X.X.X，部署了 5.1 到 5.6 三个版本的 MySQL 复制集群，版本是 Percona Server 5.1.68、5.5.30、5.6.10，每个复制集群都分别具有三个复制节点。目前主要对 5.5.30 版本进行了测试。

2.2.1.4 测试方法

开发人员在自己的开发环境构建完代码并且通过了单元测试，然后手工执行此集成测试脚本，验证上述功能模块是否能按预期的目标执行。考虑到此集成测试与数据库版本之间关系不大，目前主要是在 5.5.30 这个类似生产环境的版本的 MySQL 复制集群上测的，5.1.68、5.6.10 原来曾经测过几次，后来没有再测。

测试脚本用 BASH 和 Perl 语言开发，参考了 mysqltest 测试框架，主要是执行一些测试语句，比对实际和预期结果。

具体请参考 WIKI：集成测试项目列表

2.2.2 MySQL 兼容性测试

2.2.2.1 测试目的

检验 DBProxy 对 MySQL 语句协议的兼容程度，是否能够正确执行 MySQL 常用的语句、字符集和事务处理等。进而验证是否对应用程序是透明的。

这里只测试了关闭连接复用的情况。

打开连接复用的情况下的主要有以下支持的功能特性和使用限制：

- 支持事务
- 支持 prepare、deallocate 等预编译语句
- 支持设置当前数据库。如 use db
- 部分支持设置当前字符集。如 set character_set_results = gbk。只支持 set names、character_set_client、character_set_connection、character_set_results 四种。
- 不支持会话变量。比如 set @a = 1
- 不支持临时表，因为临时表是会话级的
- 不能很好的支持 last_insert_id()、found_rows()、SQL_CALC_FOUND_ROWS 等依赖上一条语句执行结果的函数（DBProxy 有延迟归还连接池的功能）
- 不支持 load data、load xml 等文件操作
- 不支持 LOCK、UNLOCK、flush table with read lock、get_lock()等锁相关的操作命令和函数
- 不支持 savepoint 等事务保存点相关命令
- 不支持分布式事务
- 不支持 MySQL 复制语句。比如：purge binlog、reset master 等

2.2.2.2 测试结论和分析

测试用例 310 个，全部来自 Percona Server 5.5.29 自带的 main 集成测试套件，为了和 DBProxy 兼容做了一些改写。主要包括了常用的 SQL 语句、字符集、函数和存储过程，大致分类如下：

- 查询语句：包括常用的查询和数据操作语句，比如 select，update，delete，insert 语句，以及 union，subselect，sum_distinct 语句等
- 字符集：生产环境会用到的 utf8，latin1，gbk，gb2312 都测了
- 数据定义语句：比如 alter，drop，truncate 等
- 数据类型，比如 bigint，binary，bool，bit，blob 等
- MySQL 内建函数和存储过程：func_analyse，func_compress ，func_concat ，sp-big，sp-bugs 等
- 分区表：partition_charset，partition_column，partition_column_prune 等
- 事务：commit，consistent_snapshot

测试结果举例：

```
309 disabled
310 pass
 47 skipped
```

2.2.2.3 测试环境

测试环境为持续构建服务器，是一台 Oracle Linux 6.3 服务器，IP 地址：X.X.X.X，部署了测试脚本最新版本、DBProxy 每日构建的主干最新版本、和一个后端数据库实例（Percona Server 5.5.29），客户端也用 Percona Server 5.5.29 的版本。

2.2.2.4 测试方法

MySQL 自带的 main 测试套件包含了 666 个测试用例，其中大量 DBProxy 不需要关注的，还有一些比如主从复制不需要测试、或带有不支持的语句的等，都没有执行测试，只挑选了其中 310 个执行了集成测试。

因为测试用例里带有会话变量等不被 DBProxy 支持的语句，所以测试过程中还关闭了 DBProxy 的连接复用。测试用例的数据库连接指到写端口，意味着只连接后端一台主数据库。

脚本调用 mysqltest 执行测试用例，mysqltest 比对实际测试结果和预期结果，如发现不一致便在日志里打印一条报错和较详细的错误信息。

每日构建时自动执行此集成测试，并生成测试报告。目前此集成测试是否成功不影响 nightly 版本的发布。

具体参考 WIKI：开发规范，MySQL 的集成测试

2.2.3 有压力的情况下，结果集正确性测试

2.2.3.1 测试目的

验证在较高压力和并发情况下，查询结果集的正确性。

2.2.3.2 测试结论和分析

有压力的情况下打压时间约十分钟，同时执行了几个定制的 MySQL 兼容性集成测试套件（包括四个用例，列表如下），通过了集成测试，说明 DBProxy 在较高压力（系统平均负载 4.84）、较高并发（客户端连接数 1000）的情况下，基本能保证稳定正常运行。

测试用例包括 select 查询语句和常用的字符集：
ctype_gb2312
ctype_gbk
ctype_latin1
ctype_utf8
select

2.2.3.3 测试环境

持续集成服务器，IP 地址 X.X.X.X，部署了 DBProxy 和测试脚本，和一个 Percona Server 5.5.29 数据库实例。

2.2.3.4 测试方法

一边用 sysbench 打压，一边执行定制的 MySQL 兼容性测试套件。sysbench 启 1000 个并发连接，数据量 1000000，打压时间 600 秒。mysqltest 连接 DBProxy 写端口，循环执行查询和字符集测试用例。DBProxy 关闭了连接复用，和焦点业务线的实际生产环境一致。

具体请参考 WIKI：测试是否串包？

2.3 焦点业务线的功能性测试

2.3.1.1 测试目的

模拟焦点业务线的实际的业务请求，测试通过 DBProxy 访问数据库的功能是否正常。

2.3.1.2 测试结论和分析

访问页面无异常，测试通过

2.3.1.3 测试环境

测试库 X.X.X.X
Dbproxy X.X.X.X

2.3.1.4 测试方法

由焦点业务线人员利用工具模拟实际压力对 proxy 打压，人工检查页面结果的正确性

3 稳定性测试

3.1.1 测试目的

模拟较大压力，访问 proxy，测试较长时间(>7 天)，观察 proxy 进程是否能稳定运行，另外还可以观察客户端访问的响应时间和进程占用的系统资源是否发生变化。

3.1.2 测试结论和分析

跑 sysbench 的同时，跑 select 查询和字符集查询集成测试，内存有所增长不多

3.1.3 测试环境

X.X.X.X

3.1.4 测试方法

1. 持续跑“自己通过脚本写的集成性测试”，测试过程中不要停 DBProxy。持续较长时间（约三周）。较新的版本在做其它测试时也大约稳定跑了一周。

2. 稳定性测试不需要收集响应时间，需要在较大压力的接近极限的条件下（比如 20000 个前端连接，后端 2000 个连接），跑一周时间。用 sysbench 打压。

具体请参考 WIKI：稳定性测试

4 健壮性测试

4.1 网络异常模拟测试

4.1.1 测试目的

验证在网络发生异常（主要是网络瞬间中断）的情况下，DBProxy 是否能正确处理这种情况，不会引起本身运行异常，是否能正确释放客户端和 DBProxy、DBProxy 和后端数据库的连接。

4.1.2 测试结论和分析

当前端压力较小的情况下，网络发生瞬间中断几秒钟会导致：在网络中断期间导致的 client 连接被断开。但是在连接恢复正常之后后续的插入都能够正常执行。

不过当前压力过大的情况下：即使网络没有瞬间中断都会导致 dbproxy 疲于应付，卡死(coredump)，甚至是系统 hang 住。

4.1.3 测试环境

DBProxy 处于机器 X.X.X.X；
三个 backend 分别是 X.X.X.X:3401、X.X.X.X:3402、X.X.X.X:3403
测试 client 在 X.X.X.X

4.1.4 测试方法

丢包包括 client 端到 dbproxy 的数据包及 dbproxy 到后端 backend 端的数据包丢失。
因而下面会测试两种场景：dbproxy 与 mysqlserver 间的网络的瞬断及 client 与 dbproxy 的网络瞬断。

客户端执行自己通过脚本写的集成测试，后端在 X.X.X.X 上面间隔性的执行 iptables 命令通断网络，前端在 X.X.X.X 上面执行查询语句脚本，并比对查询结果。

具体请参考 WIKI：网络中断模拟

5 性能测试

5.1 焦点的性能衰减测试

5.1.1 测试目的

5.1.2 测试结论和分析

本次测试中未出现’假死‘现象，但是在一次尝试测试中，dbproxy 出现 crash，需开发组同学进一步验证。
本次测试 dbproxy 在 qps 上性能有明显提升。
每秒创建的连接数据对比 （mysql：dbproxy）：（ 8718：278）。复用比例较高，减少 db 端创建连接的资源消耗。

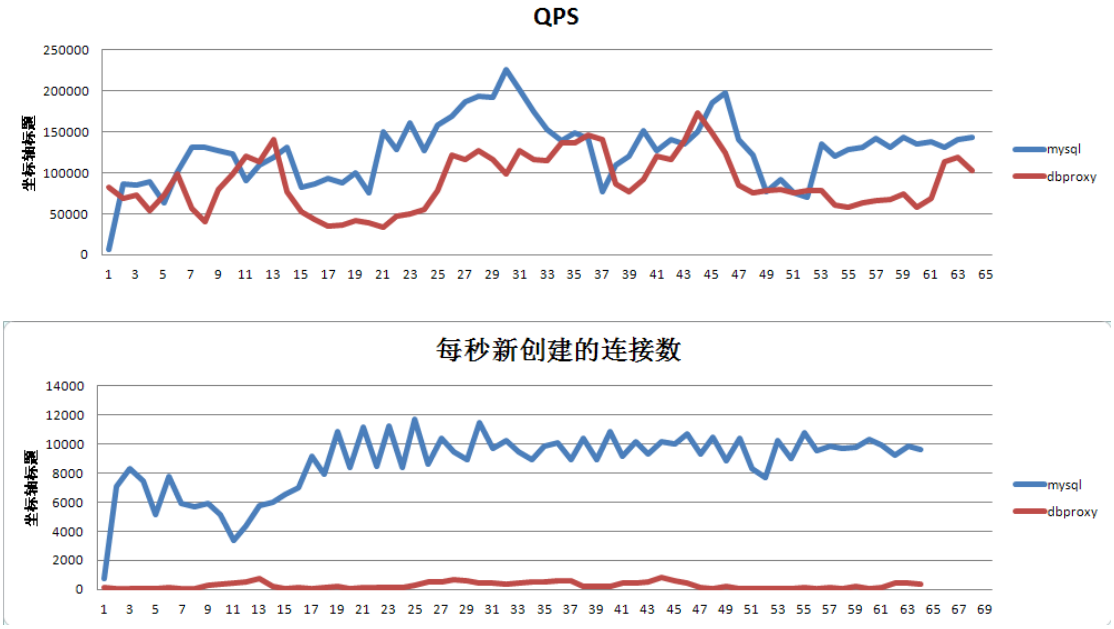
5.1.3 测试环境

测试库 X.X.X.X
Dbproxy X.X.X.X

5.1.4 测试方法

测试目的
1.结合业务线压力测试 dbproxy
2.验证 dbproxy 连接复用系能性能
3.验证 dbproxy 对 sql 的性能影响

前端发起 500 并发（比较接近业务真实压力），尝试 1000 并发，打压工作报错。
压力测试时间为 1 个小时。



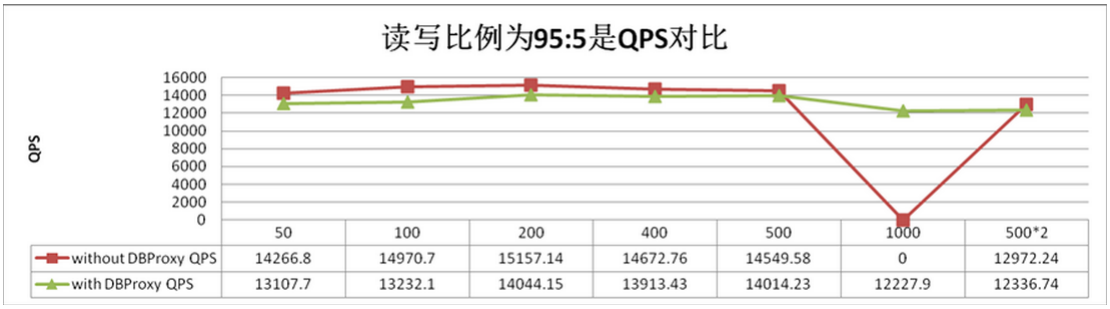
具体请参考相关邮件

5.2 ycsb 性能衰减测试

5.2.1 测试目的

测试通过 DBProxy 产生的性能损耗。具体测试指标是吞吐量（QPS）、响应时间。

5.2.2 测试结论和分析



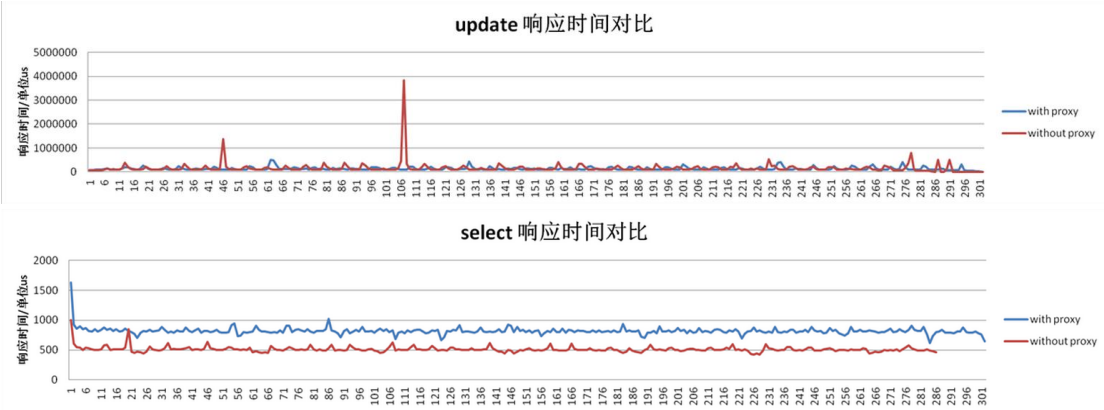
	50	100	200	400	500	1000	500*2
without DBProxy QPS	14266.8	14970.7	15157.14	14672.76	14549.58	0	12972.24
with DBProxy QPS	13107.7	13232.1	14044.15	13913.43	14014.23	12227.9	12336.74

说明： 图中直接打压 mysql 的性能突然降低是因为单个 YCSB 启 1000 个线程，打压 mysql 时会卡住，没有结果输出。并不是 mysql 的 qps 为 0

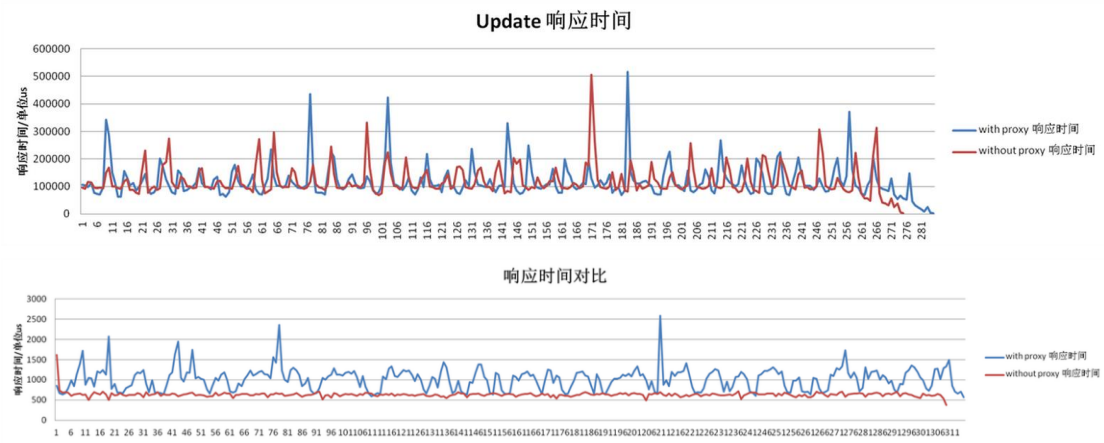
在读写比例为 95:5 或者读更多的情况下，DBProxy 相对来讲会有些许的性能损耗，单损耗不大在 10%以内。

100 并发，读写比例为 50:50 时响应时间对比图

由于 dbproxy 是 cpu 和网络 io 消耗性的，这时 2 核 16 个线程的 cpu 使用率大约 80%，其中 60%在消耗在用户态（user%），40%消耗在内核态(sys%)。



100 并发，读写比例 95:5 的响应时间对比



1. DBProxy 的 Update 的响应时间基本没有太大的波动，直接打 mysql 类似。
2. Select 的响应时间反差较大。经过 dbproxy 以后，响应时间变长，波动变大。（与之前 jemeter 测试结果不符）

5.2.3 测试环境

Client:
单独的物理机 cpu Intel(R) Xeon(R) E5-2650 0 @ 2.00GHz
mem126G
disk SAS 3T
net Intel Corporation I350 Gigabit

DBProxy 及 PXC:
DBProxy 后端挂载 1 个 Backend，数据库服务；DBProxy 和 mysql server 分别部署在一台单独的物理机上面；(Dbproxy:X.X.X.X;PXC：X.X.X.X)
物理机 cpu Intel(R) Xeon(R) E5-2650 0 @ 2.00GHz
mem126G
disk SAS 3T
net Intel Corporation I350 Gigabit

PXC5.5.30-log Percona XtraDB Cluster (GPL), wsrep_23.7.4.r3843
Dbproxy 0.7.0
ycsb

数据量 100 万

5.2.4 测试方法

前端采用 ycsb 直接打 mysql-server 和经由 dbproxy 的性能的对比

具体请参考相关邮件

5.3 测试组 jmeter 的性能衰减测试

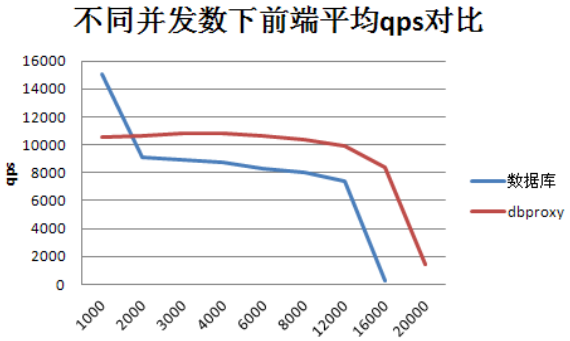
5.3.1 测试目的

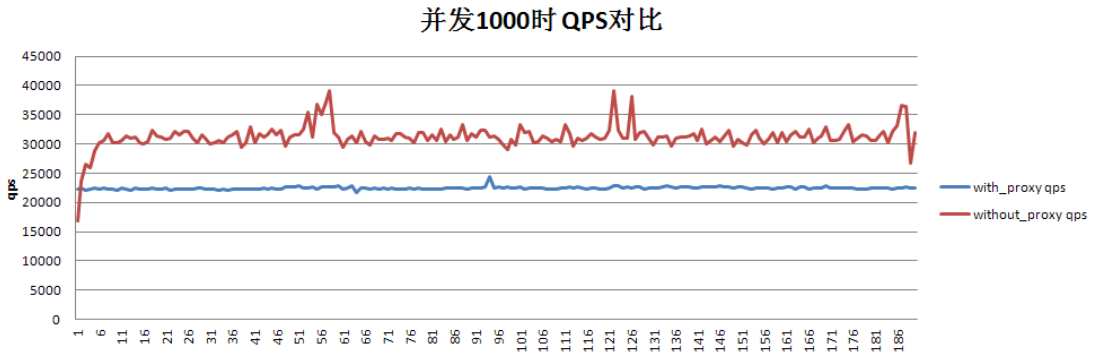
第三方测试组用 jmeter 以较高并发对 proxy 打压，测试大压力下 proxy 性能损耗。

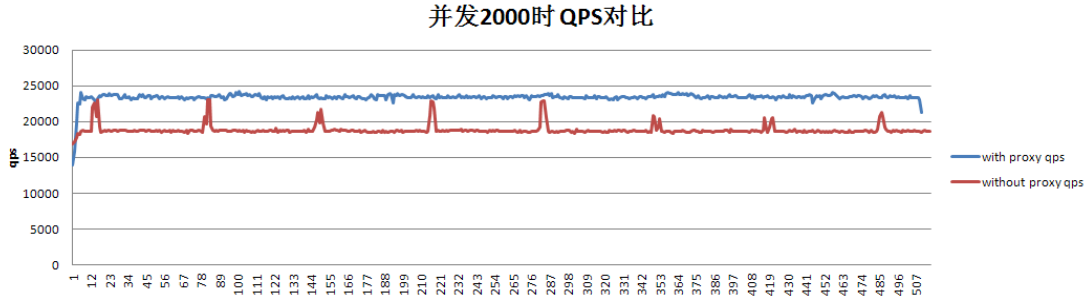
5.3.2 测试结论和分析

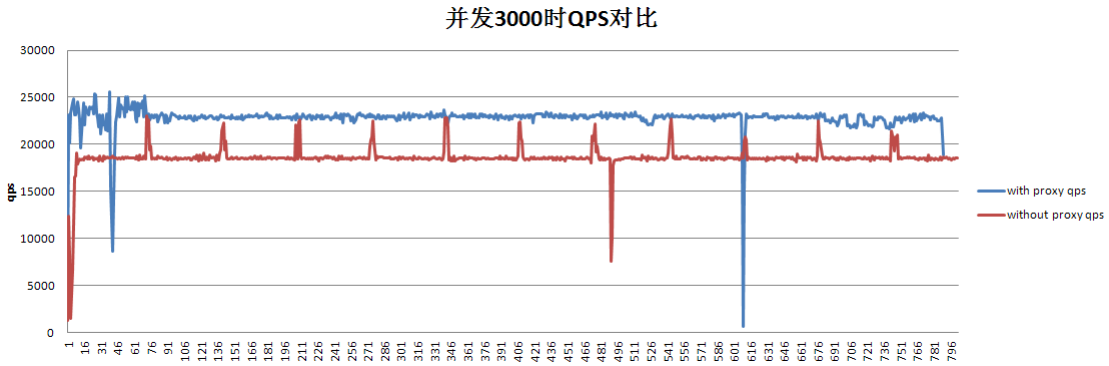
经过较长时间的折腾，与测试组的测试结果基本出来了。结论是：

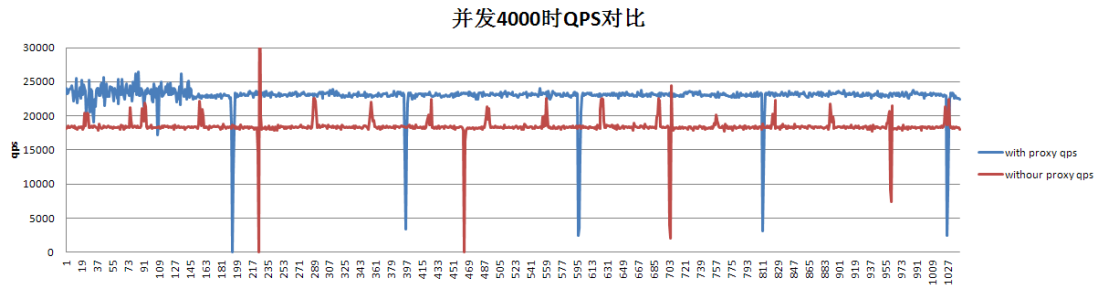
1. 从后端看，与直连数据库相比 dbproxy 的性能损耗较小，相反在大并发下 dbproxy 的性能会较好；
2. 随着并发数的增大，并发数大于 1000 时，直连数据库和经由 DBProxy 都会导致 QPS 下降，但是经由 DBProxy 的 QPS 下降较慢；
3. 随着并发数的增大，每秒 QPS 抖动也逐渐增大；
4. 因为数据误删除。。。。前端响应时间的数据暂时没有

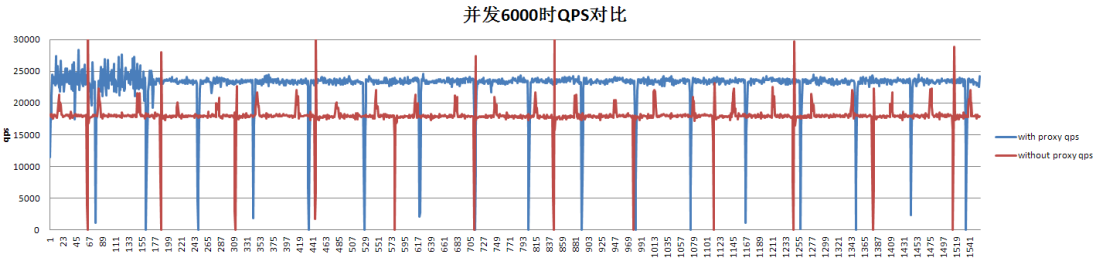


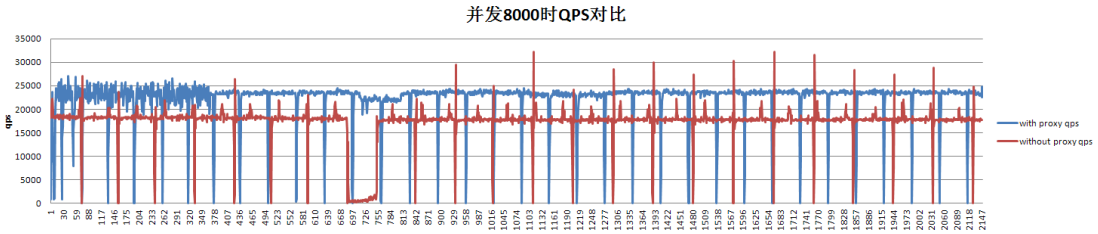


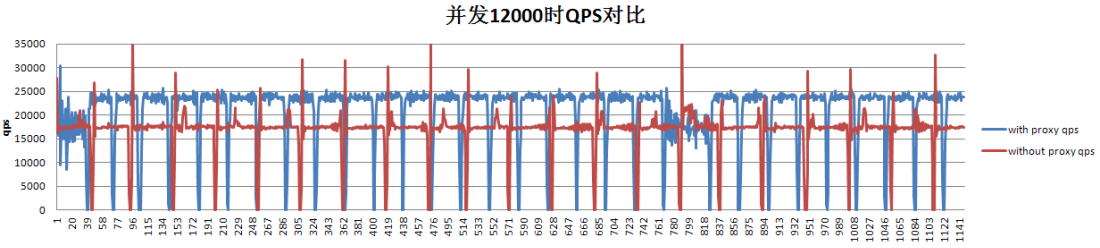


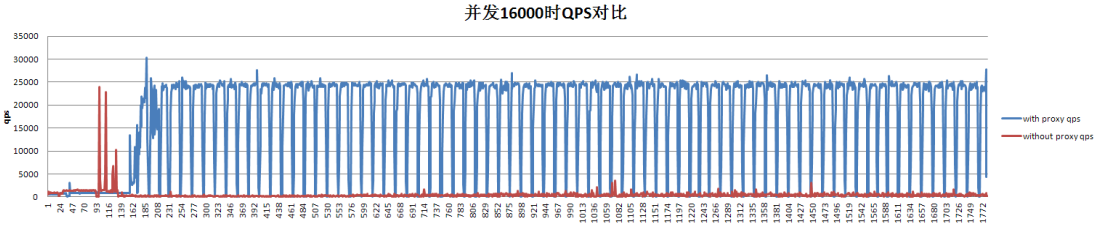




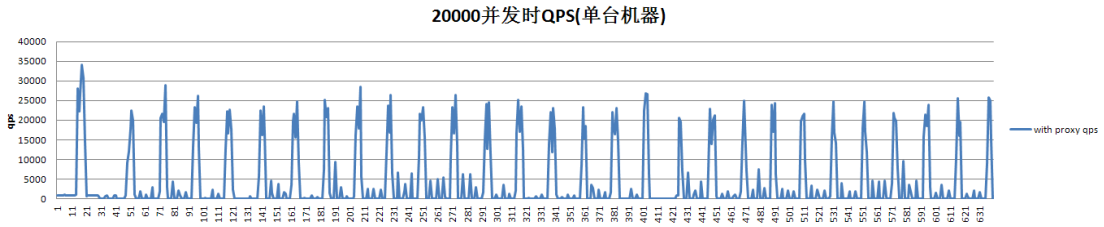








并发 2000 时，DBProxy 的 QPS，直连数据库 qps 很低没有记录



5.3.3 测试环境

DBProxy 后端挂载 1 个 Backend，数据库服务；DBProxy 和 mysql server 分别部署在一台单独的物理机上面；(Dbproxy:X.X.X.X;PXC：X.X.X.X)

Dbproxy 及 PCX 端：
物理机 cpu Intel(R) Xeon(R) E5-2650 0 @ 2.00GHz
mem126G
disk SAS 3T
net Intel Corporation I350 Gigabit

PXC5.5.30-log Percona XtraDB Cluster (GPL), wsrep_23.7.4.r3843
Dbproxy 0.7.0_20130906(多线程版本)
Jemeter

5.3.4 测试方法

前端采用几个 jmeter client 同时加压，测试并发到达 1000 到 2000 时，直接打 mysql-server 和经由 dbproxy 的性能的对比。

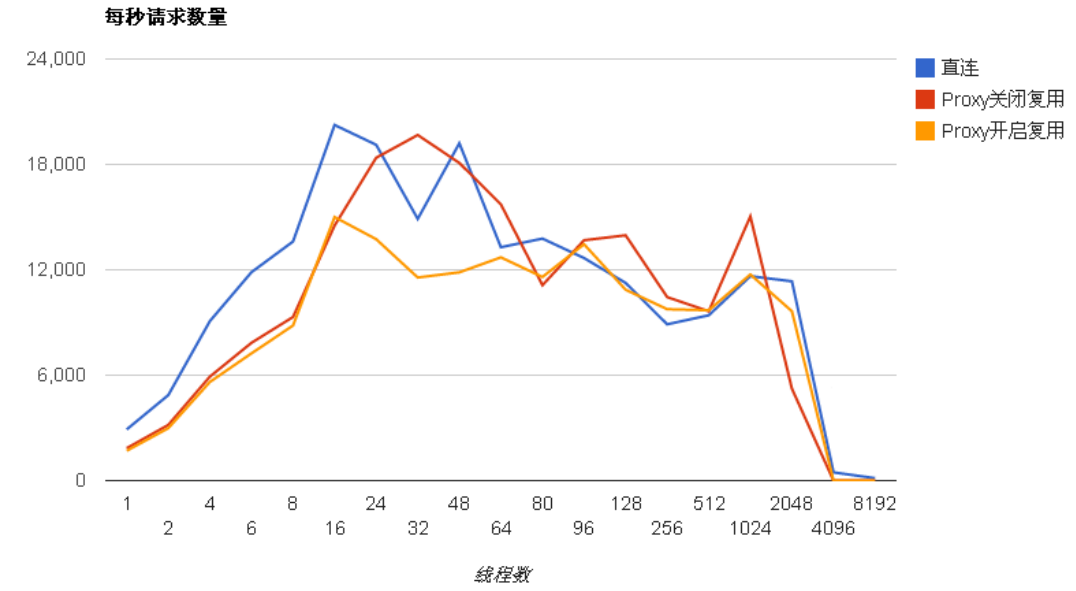
5.4 性能基准测试

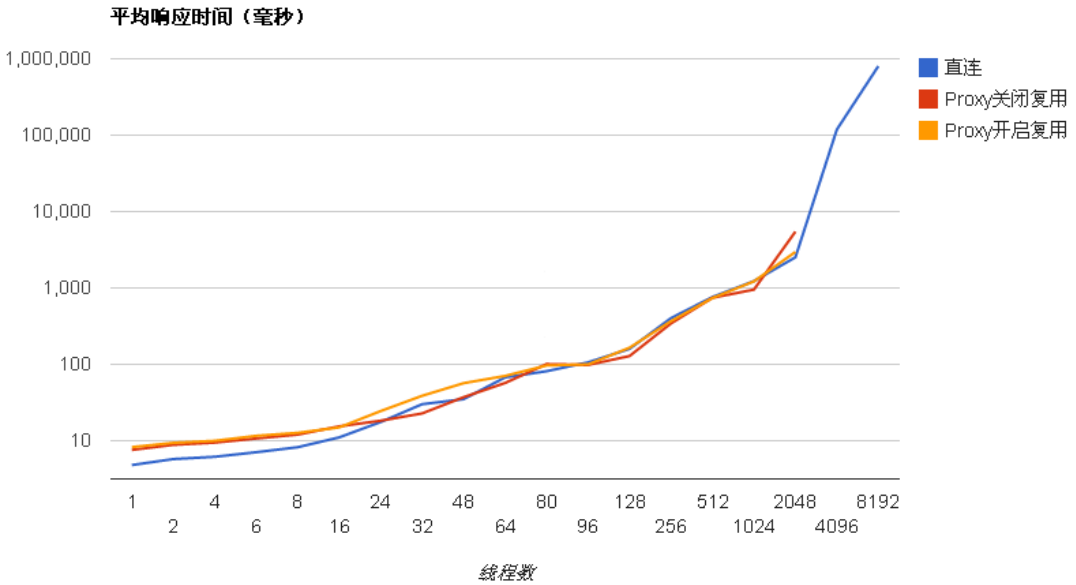
5.4.1 测试目的

测试随并发数增长，对 DBProxy 的响应时间和吞吐量的性能测试，期望目标是应该是一个线性增长的趋势。

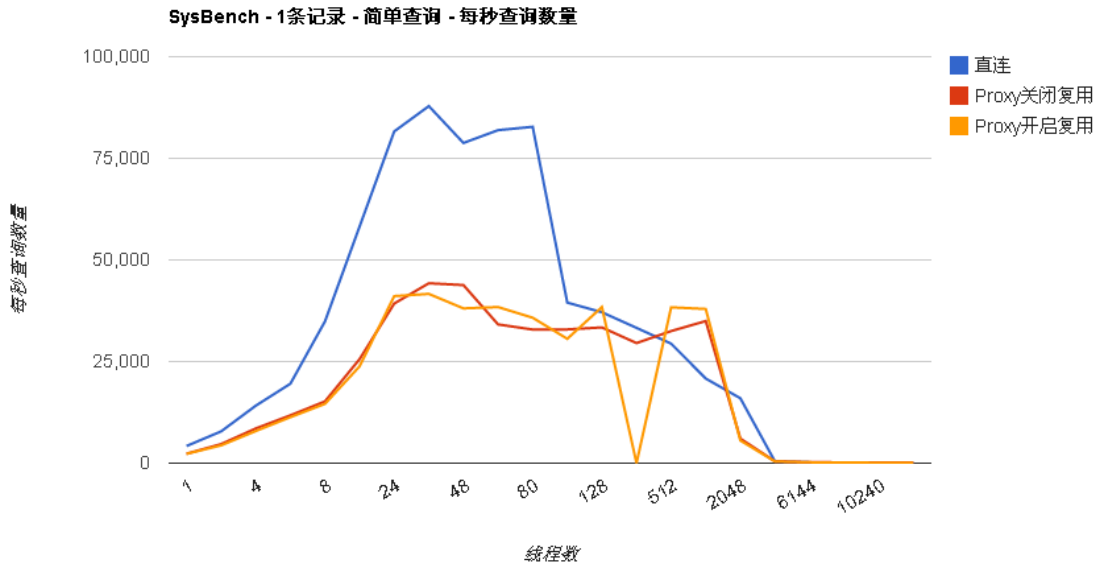
5.4.2 测试结论和分析

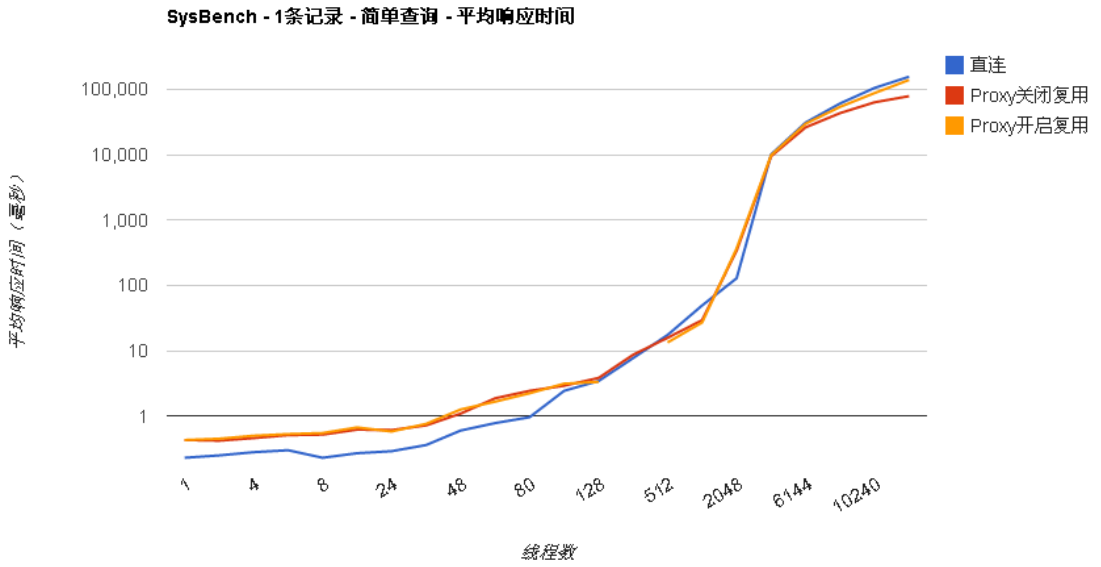
5.4.2.1 SysBench，10M 记录，复杂查询





5.4.2.2 SysBench，1 条记录，简单查询

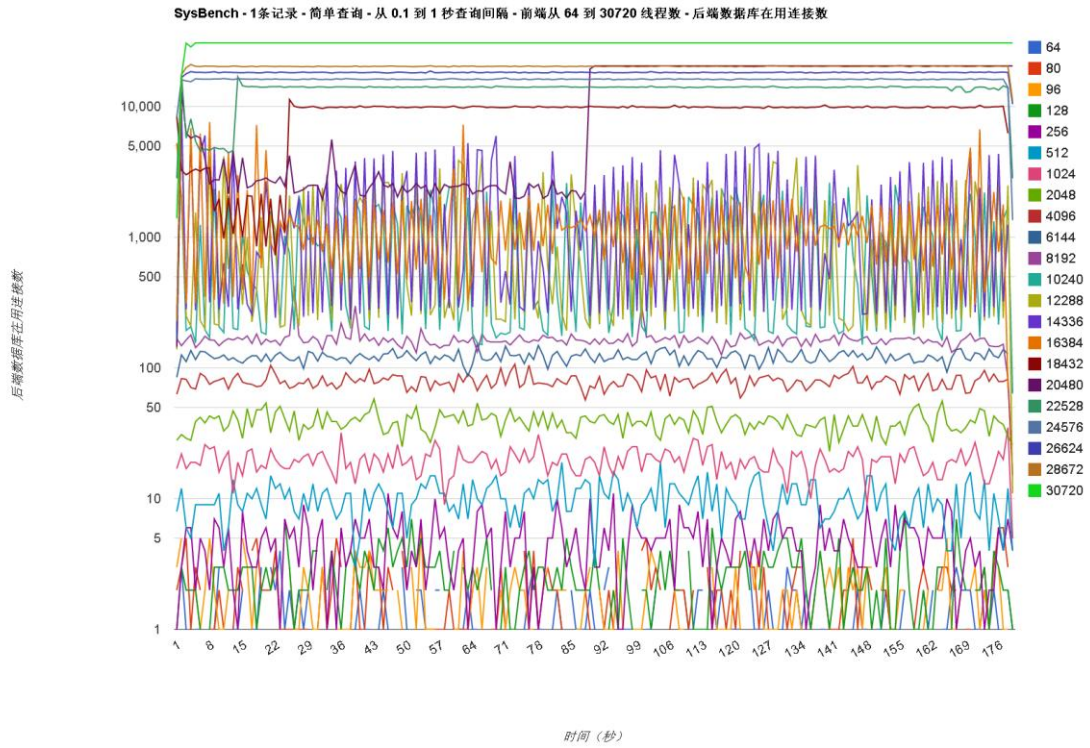


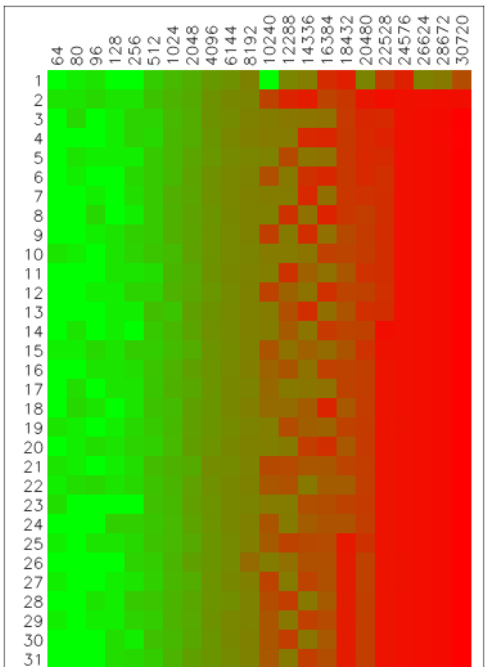


1. 一开始 QPS 随并发数线性增长；当超过 CPU 个数后，性能下降；较高并发数 2048 之后，直线下降（Proxy，MySQL 系统负载都较高）。响应时间基本是随并发数上升上涨。
2. QPS 性能有时有波动，原因不清，可能和外界环境影响有关。
3. 通过 Proxy 对比直连 MySQL 性能下降 25%-50%左右。
4. Proxy 开启对比关闭复用性能有轻微下降。
5. 通过 Proxy，并发 4096,8192，Sysbench 执行失败。原因是 DBProxy 后端检测超时，修改数据库状态有问题，已修改程序。

需要启用 MySQL 的 thread pool 再测一遍

5.4.2.3 连接池的使用情况（SysBench，简单查询，前端连接 64 到 30000）

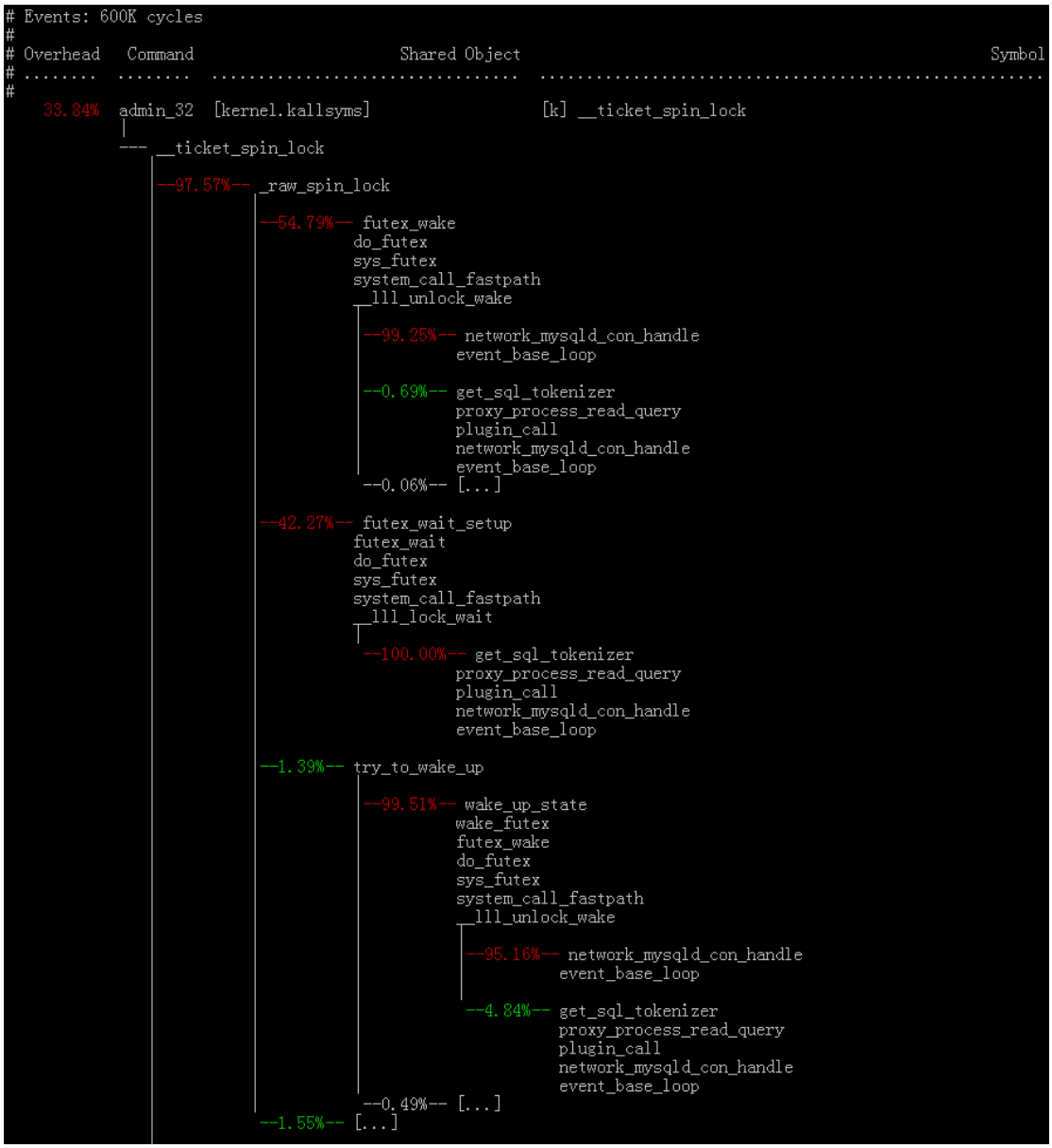




1. 前端少于一万个连接时，复用效果较好。超过一万，波动较大。超过两万，基本就没法用了
2. 随前端连接数增加，查询响应时间也增长，导致连接复用越来越差

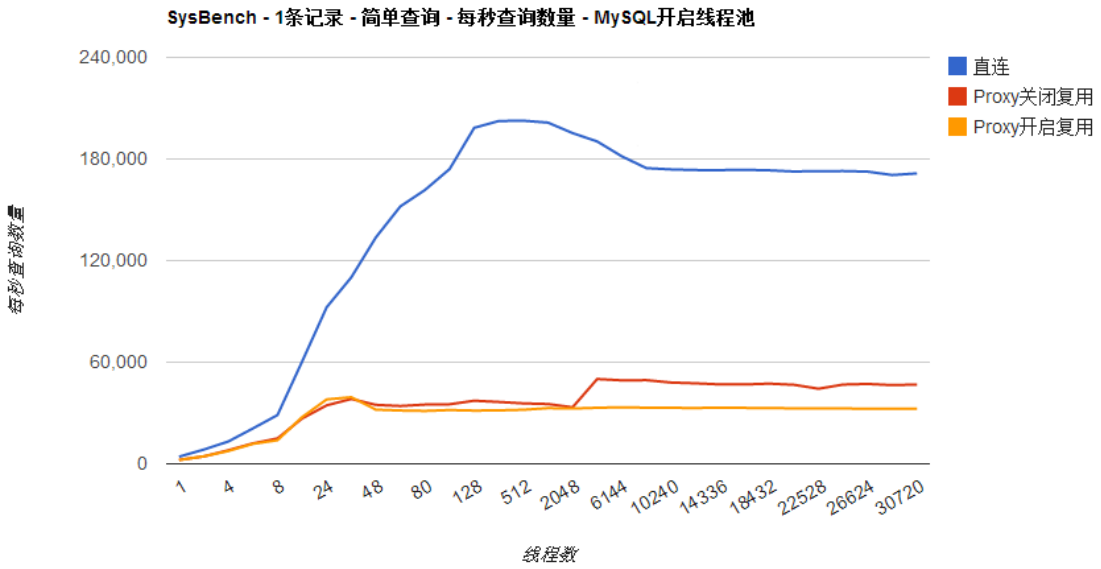
同上，需要启用 MySQL 的 thread pool 再测一遍。

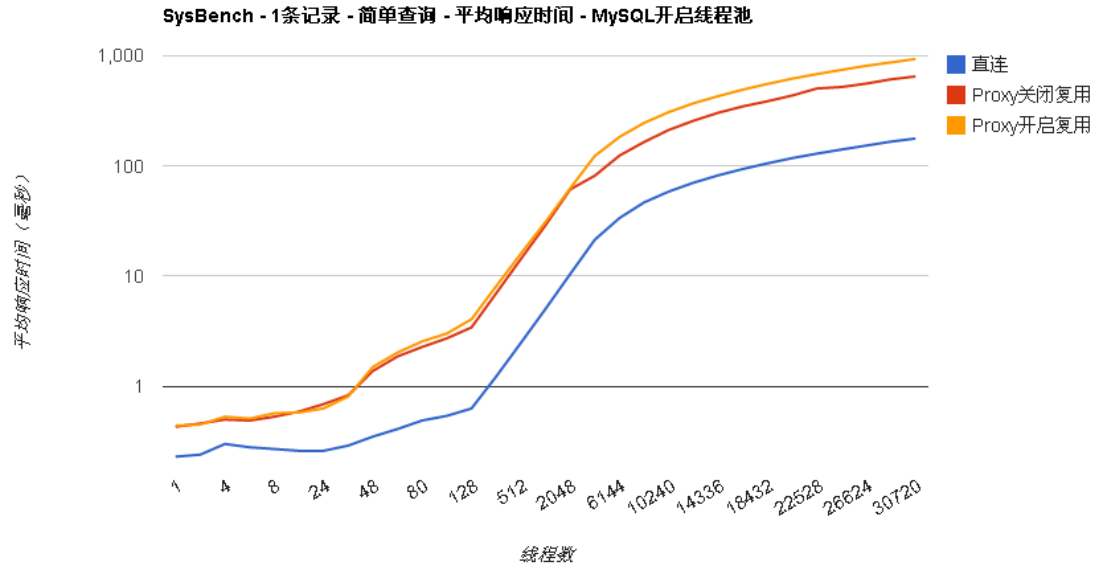
5.4.2.4 查看最占 CPU 时间的系统调用（SysBench24 线程打压）



貌似 get_sql_tokenizer 互斥锁占用了较多 CPU

5.4.2.5 打开 MySQL5.5 线程池后再打压（SysBench，1 条记录，简单查询）





24 个线程 proxy 基本上就达到了最大 QPS 值，40K，性能衰减 50%，之后 QPS 比较稳定。直连 QPS 还继续上升， 512 线程到达最大值，200K，保持稳定，proxy 和直连相比性能衰减 80%。开启复用比关闭复用的 QPS 性能衰减约 10%-30%。
proxy 响应时间是直连的 2-6 倍，当 30K 线程时接近 1 秒。

由于 dbproxy 是 cpu 和网络 io 消耗性的。
连接数在 128 时，这时 2 核 16 个线程的 cpu 使用率大约 75%，其中 25%在消耗在用户态（user%），50%消耗在内核态(sys%)。

连接数超过 6000 时，这时 2 核 16 个线程的 cpu 使用率大约 95%，其中 30%在消耗在用户态（user%），70%消耗在内核态(sys%)。

5.4.3 测试环境

3 台物理机，配置 32 CPU、128G。Sysbench、Proxy、MySQL 分别用一台。
X.X.X.X sysbench-0.4
X.X.X.X Proxy
X.X.X.X Percona-Server-5.5.29

5.4.4 测试方法

用 Sysbench 打压，测试从 1 到 30K 并发连接数情况下，通过 DBProxy 和直连 MySQL 的吞吐量和响应时间。Sysbench 测试的记录数 1 条，测试只读查询。MySQL 缓存大小 4G，打开线程池管理。

具体请参考 WIKI：性能基准测试

6 测试问题

1. 某业务线测试，Proxy 宕掉过一次的，后来第二遍测试没有重现。因为没有设置 coredump，无法查原因。已修改程序打开 coredump，以后再重现就能查了。
2. 集成测试，执行脚本 test-19-x 内存增长较明显，每执行一次涨 xxxK。长期测试，跑 4 天内存到 88M，最后一天观察长了约 1M。增长不是太多，影响不大，原因待查。
3. 性能基准测试，perf 查看到 network-mysqld-con-handle 和 get_sql_tokenizer 占 CPU 较多