Table of Contents

MySQLProxy性能测试报告

- 测试目的
  - 测试工具
- 测试环境
  - 硬件环境
    - 测试环境(老硬件)
    - 在线环境(新硬件)
  - 网络环境
    - 测试环境网络
    - 在线环境网络
  - 软件环境
  - 软件配置
    - 测试数据库相关信息
      - sysbench数据库初始化信息
- sysbench测试数据
  - 测试数据介绍
    - 测试场景
    - 测试维度
    - 待测试软件介绍
    - 测试脚本
      - 测试环境测试脚本
  - 详细测试数据
    - 多软件对比
      - 数据对比图
    - 新机器10分钟事务测试
      - 数据对比图
    - 新机器10分钟读写分离测试
      - 数据对比图
  - 数据分析
    - 单proxy性能分析
      - 单机处理能力
      - 延迟分析
      - 多proxy集群部署
      - 多集群的proxy扩展性
    - 带宽瓶颈
    - 数据库性能瓶颈
    - proxy与友商软件的对比
    - 读写分离的表现
- tpcc测试数据
  - 测试数据介绍
    - 测试场景
    - 测试维度
    - 测试软件配置
  - 测试数据
    - 数据对比图
    - 数据分析
- 附录:

软件的配置信息
MySQL(master)
MySQL(slave)
proxy(final-single)
keepalived配置
测试环境测试脚本
在线环境测试脚本
读写分离测试用lua

MySQLProxy性能测试报告

编写人：何伟 版本 1.0.1 修改日期：2016.09.01

测试目的

本次测试主要是为了测试我们自身在MySQLproxy基础上改进开发的数据库中间件(以下简称proxy)在线上硬件环境下的性能，着重对比通过proxy访问与直连MySQL的性能对比情况，同时基于proxy的限制，以及后续可能的实际部署情况，我们也按照实际可能的部署方式进行了部署。验证了在实际部署情况下的性能参数，为后续系统的上线部署和相关性能参数评估提供了一定的依据。

测试工具

本次测试使用了两种测试工具，sysbench和tpcc-MySQL(以下简称tpcc)，其中，sysbench用来测试在压力情况下的proxy的性能表现。tpcc用来测试在模拟线上业务的情况下，proxy能支持到的业务情况。

测试环境

本次测试主要在杭州BGP机房的测试用服务器上进行，评估在系统真实线上应用时的性能数据，同时，针对各个不同的代理软件，我们也在较老的硬件上的进行了一些简单的测试，用来做一个不同平台的性能比较。

硬件环境

测试环境(老硬件)

Lenovo R520 * 3 Intel Xeon 5130 @ 2.0GHz * 2(4 cpu core at all); 4G DDR2 667 * 2(Dual Channel); 1T 7.2k sata

Dell 2950 * 1 Intel Xeon 5405 @ 2.0GHz * 2(8 cpu core at all); 4G DDR2 667 * 4 (Dual Channel); 400G 10k sas * 2 (raid 1)

其中，老环境的测试硬件Dell2950因为IO能力有限，所以相关的数据都放到了内存里。减少了IO对系统性能的影响。

在线环境(新硬件)

Dell R730*2 Intel Xeon E5-2650 v3 @ 2.30GHz * 2(40 cpu core at all); 256G DDR3(Dual Channel); 4 * 1T(raid 5)

Dell R730*2 Intel Xeon E5-2650 v3 @ 2.30GHz * 2(40 cpu core at all); 128G DDR3(Dual Channel); 2 * 1T(raid 1)，2 * 800G SSD(raid 1)

网络环境

测试环境网络

业务类型	IP地址	服务器类型	服务器操作系统
sysbench	host1	Lenovo R520	Ubuntu14.04.1
lvs	host2_1(host2_2)	Lenovo R520	Ubuntu14.04.1
proxy	host3_1(host3_2)	Lenovo R520	RHEL6.5
DB	host4	Dell 2950	RHEL7.2

测试各个服务器均直连到同一上行交换机上，其中lvs与proxy之间有一根直连线路（lvs NAT 需要不同网段）。

各个网卡均是千兆全双工，线路质量稳定。

在线环境网络

业务类型	IP地址	服务器类型	服务器操作系统
lvs+sysbench	host5	R730	RHEL7.2
proxy	host6	R730	RHEL7.2
DB(online/master)	host7	R730	RHEL6.6
tpcc(host8_1)	host8_2	R730	RHEL6.6

在线环境两个服务器在一个机架上，部署在萧山机房，数据库在自身网段，部署在滨江机房。

业务服务器部署了3网卡内网bonding，数据库服务器是千兆直连。

各个网卡均是千兆全双工，线路质量较好，滨江到萧山的常规延迟在0.2ms左右。

软件环境

测试软件	版本	备注
sysbench	0.5	自己修改了一个特定的读写分离的测试用lua，新增了部分参数
MySQL	5.6.31	当前最新的5.6分支社区版本，仓库RPM安装(测试环境)
MySQLproxy	6da5405b(final分支)	7月27日最后提交的。这里主要用来评估系统
lvs	1.2.13(ubuntu仓库)	使用的NAT方式，使得一个机器上不同端口部署多个proxy
mycat	1.5.1-RELEASE-20160811220036	使用的最新的稳定版
oneproxy	rhel6-linux64-v5.8.4-ga	官方提供的rhel6的二进制发行包
MySQL	5.7.14	在线环境
tpcc-mysql	faa06df@master	https://github.com/Percona-Lab/tpcc-mysql

软件配置

由于硬件数量的限制，我们的测试环境数据库主从均运行在同一台服务器上。为了减少IO对性能的影响，提升数据库的处理能力，我们把主从数据库的数据文件均配置到系统内存中。减少了IO的开销和IO等待导致的性能下降，提升了数据库的最大处理能力，由于硬件有限，我们没有把主从服务器分开部署，所以，数据不一定非常准确，本报告的性能数据主要是用作一个量级的参考。

由于能力有限，这里我们把测试软件的配置都详细记录下来，后续好做进一步的性能分析。

在线环境的配置是MySQL主从独立部署。均有SSD盘储存数据文件。

MySQL和lvs，以及proxy的配置信息详见附录。

测试数据库相关信息

sysbench数据库初始化信息

测试数据库我们是使用sysbench自己创建的对应测试库的。建库的语句如下：

```
sysbench --test=/home/mysql-cetus/sysbench/sysbench/tests/db/oltp.lua --oltp-table-size=100000 --mysql-table-engine=innodb --mysql-user=test --mysql-password=xxxxxx --mysql-port=3306 --mysql-host=host4 --mysql-db=oltp --max-requests=0 --max-time=180 --oltp-tables-count=10 --report-interval=$INTERVAL --num-threads=50 prepare
```

在线环境的sysbench初始化语句与测试一致。

tpcc数据库初始化信息

我们使用了tpcc自身的初始化工具进行了数据库的初始化。tpcc测试使用的是同一个数据库进程的上的不同的database，本次测试中，我们新建了100个warehouse。

```
mysql -utest -pxxxxxx tpcc -hhost8_1 <create_table.sql
bash ./load_warehouse_para.sh tpcc 100 10

echo "mysql -utest -pxxxxxx tpcc -hhost8_1 <add_fkey_idx.sql"
```

其中，我们是等待数据都加载完之后。再导入的索引和外键相关的依赖。

load_warehouse_para.sh是我们在官方并行导入基础上做了小修改的版本，提升了在小仓库数上的并行度。减少了导入时间。

sysbench测试数据

测试数据介绍

测试场景

根据业务的需求，我们暂时选取了两种情境进行测试。

1. 纯事务场景

   本场景中，所有的操作都是在事务里进行处理。每一个事务包括十条左右的简单查询，4条带分组，排序，统计，唯一值等条件的较复杂查询语句，然后是一条删除，一条修改，一条新增记录。加上事务开始和结束标识，一共有19次交互。

2. 读写分离场景

   根据业务场景，我修改了测试用的lua脚本，将前十三个简单查询的放到事务外，以便中间件进行读写分离操作，通过请求的读写分离扩展系统的总容量，将一个带唯一值的处理和后续的3个更新操作放到一个事务中进行处理，一次完整的测试和数据库有19次交互。

   测试本身的数据集不大，数据总量在2G左右，两种测试场景测试之前，均多次跑过类似的测试，保证数据充分预热。

测试维度

本次测试我们主要考虑测试proxy在各个线程情况下的工作性能，主要包括每秒事务数，每秒请求数，平均响应时间以及95%请求成功响应的响应时间。

为了对各个线程情况下的状态进行测试，我们从1个线程开始，以4倍递增，在超过100以后，以100开始，按两百递增，基本保证800线程的测试数据完整。不保证1600线程的测试能完整运行（测试程序对环境要求较高。有sql报错基本都会退出测试，导致测试没有结果）。

前期测试带有验证性质，每次测试没有只需太长时间。基本每个测试运行10分钟。用于简单评估各个软件版本之间的性能特性的比较测试测试用时也是10分钟。

待测试软件介绍

本次测试，我们主要关心部署了proxy之后，和直连myql直接的性能差异，同时，也通过lvs+proxy这种更接近线上部署的方式来确认lvs会给系统带来多大延迟，以及多个proxy之间的容量扩展是否线性。

后期，出于简单性能对比的目的，对友商的oneproxy和mycat进行了部署和简单测试，由于友商的软件操作不熟练，部分程序配置可能不是最优。相关测试数据只代表测试用配置下的工作表现。

测试脚本

测试环境测试脚本

其中读写分离脚本是使用修改后的lua测试脚本。名字是oltp_weihe.lua,做读写分离测试时，对应修改脚本中的lua文件名即可。其他的未做调整。

详细测试脚本和修改后的lua文件见附录中的测试脚本项。

详细测试数据

多软件对比

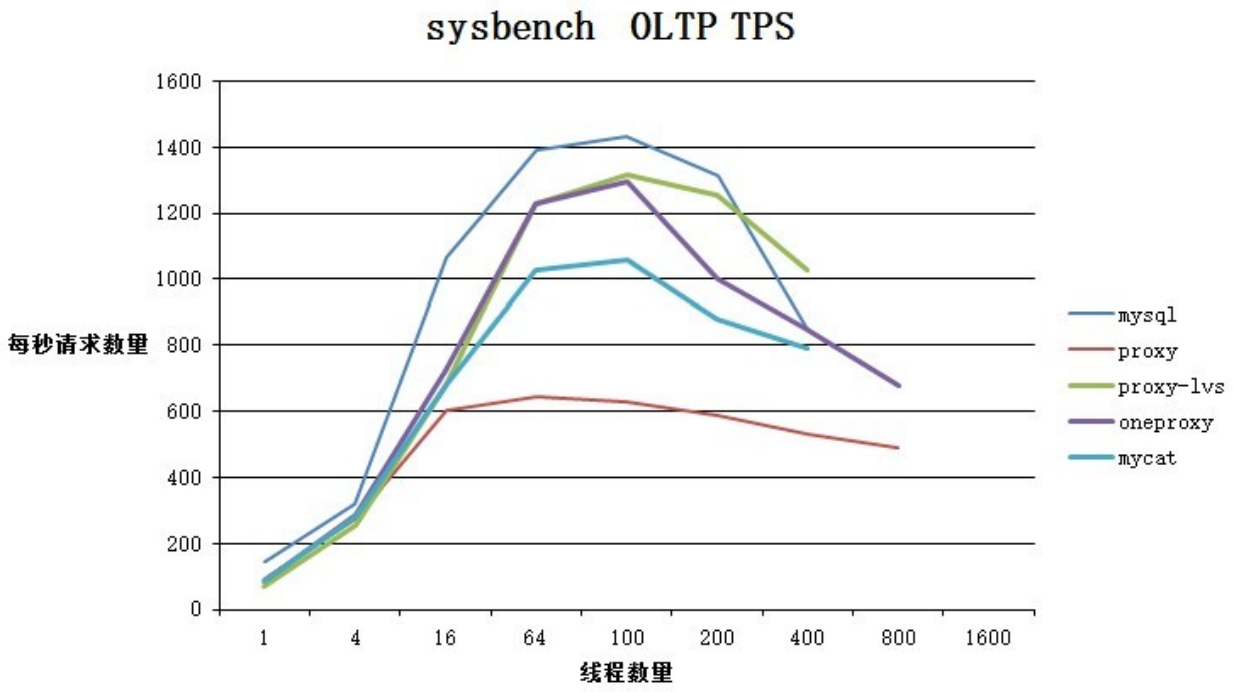
本次主要是测试各个软件的性能对比。实在测试环境上做的，跑的时间比较短，主要用来体现各个软件版本之间的相对差异。

MySQL	transaction/sec	request/sec.	reponse/ms	avg95/ms	cpu	tps/cpu
1	144.67	2604.11	6.91	7.09	0.00	0
4	319.44	5749.91	12.52	20.11	0.00	0
16	1064.57	19162.29	15.03	29.53	0.00	0
64	1391.12	25040.16	45.99	103.35	0.00	0
100	1432.01	25776.18	69.82	158.33	0.00	0
200	1314.5	23661.24	152.07	323.60	0.00	0
400	846.04	15228.70	472.51	848.69	0.00	0
800	680.82	12254.79	1174.26	1997.21	0.00	0
proxy(single)	transaction/sec	request/sec.	reponse/ms	avg95/ms	cpu	tps/cpu
1	90.56	1630.05	11.04	11.38	17.00	532.71
4	275.27	4954.86	14.53	17.63	47.50	579.52
16	601.43	10825.79	26.60	33.59	94.00	639.82
64	643.89	11590.05	99.37	110.42	100.00	643.89
100	630.75	11353.45	158.46	174.82	100.00	630.75
200	589.04	10602.66	339.01	359.89	100.00	589.04
400	532.73	9589.11	749.09	782.33	100.00	532.73
800	490.75	8833.50	1622.53	1755.47	100.00	490.75
proxy(lvs)	transaction/sec	request/sec.	reponse/ms	avg95/ms	cpu	tps/cpu
1	72.01	1296.16	13.88	16.47	14.00	514.36
4	255.42	4597.59	15.66	18.13	50.00	510.84
16	676.90	12184.24	23.63	32.09	140.00	483.50
64	1231.26	22162.73	51.96	95.70	260.00	473.56

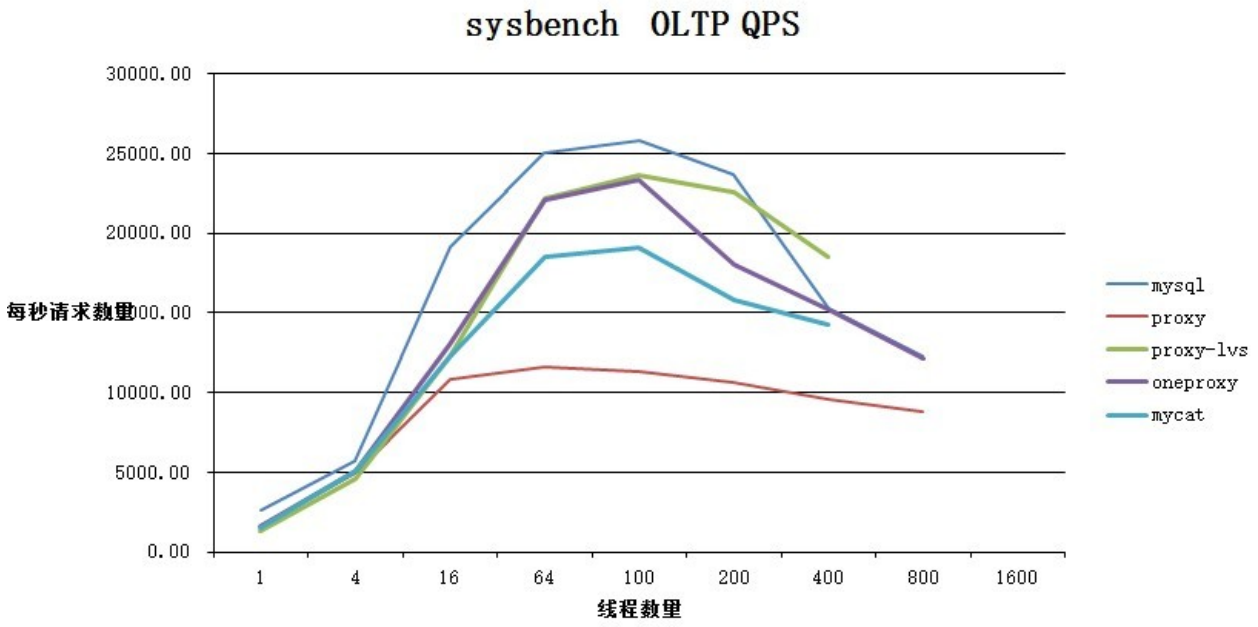
100	1315.05	23670.82	75.97	152.75	270.00	487.06
200	1253.27	22558.94	159.52	325.45	260.00	482.03
400	1026.28	18473.05	389.45	754.73	220.00	466.49
oneproxy	transaction/sec	request/sec.	reponse/ms	avg95/ms	cpu	tps/cpu
1	86.56	1557.99	11.55	12.03	22.30	388.16
4	283.88	5109.78	14.09	16.78	70.00	405.54
16	724.52	13041.43	22.08	31.82	165.00	439.10
64	1229.31	22127.50	52.04	95.07	265.00	463.89
100	1295.79	23324.24	77.14	149.71	280.00	462.78
200	999.63	17993.39	199.97	359.99	230.00	434.62
400	847.16	15248.89	471.89	832.59	200.00	423.58
800	675.34	12156.32	1183.73	1956.96	170.00	397.26
mycat	transaction/sec	request/sec.	reponse/ms	avg95/ms	cpu	tps/cpu
1	84.63	1523.42	11.81	12.33	29.90	283.04
4	676.89	12183.94	23.63	30.85	245.00	276.28
64	1027.85	18501.28	62.23	90.44	360.00	285.51
100	1057.79	19040.19	94.48	138.79	380.00	278.37
200	878.86	15819.47	227.42	363.24	320.00	274.64
400	790.42	14227.64	505.22	839.85	290.00	272.56

数据对比图

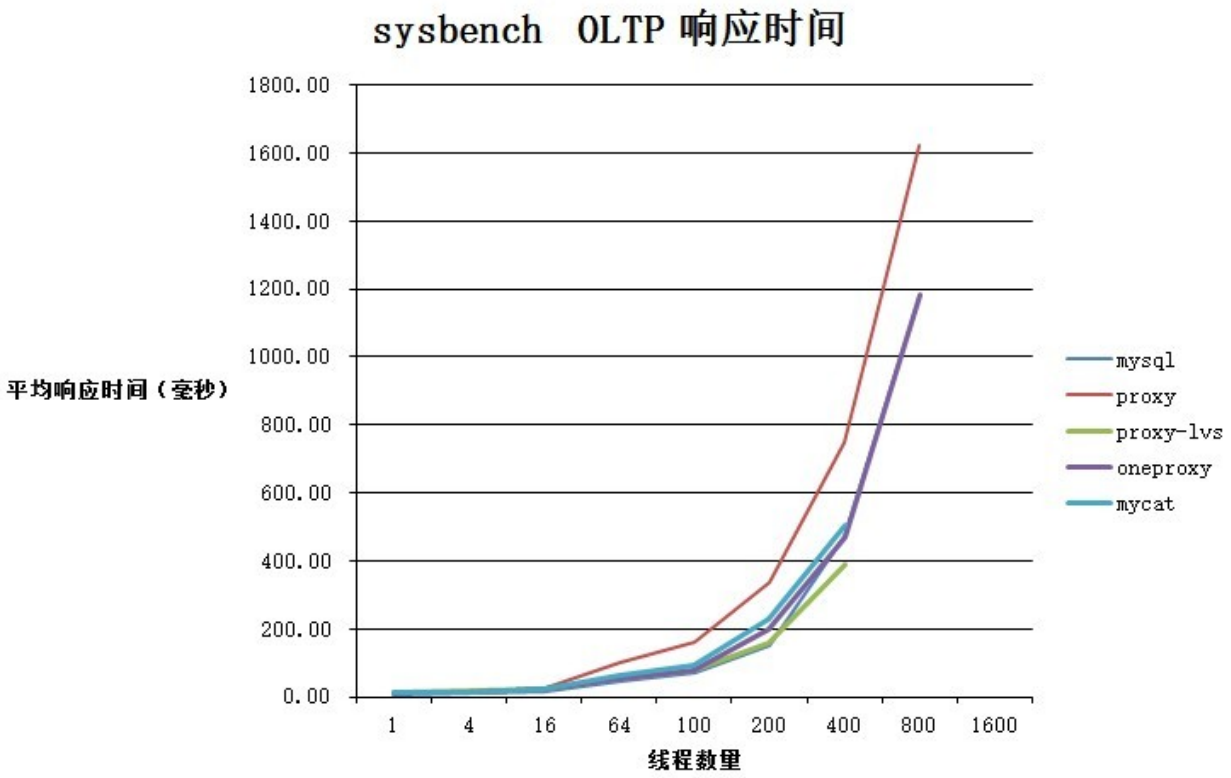
- tps对比图



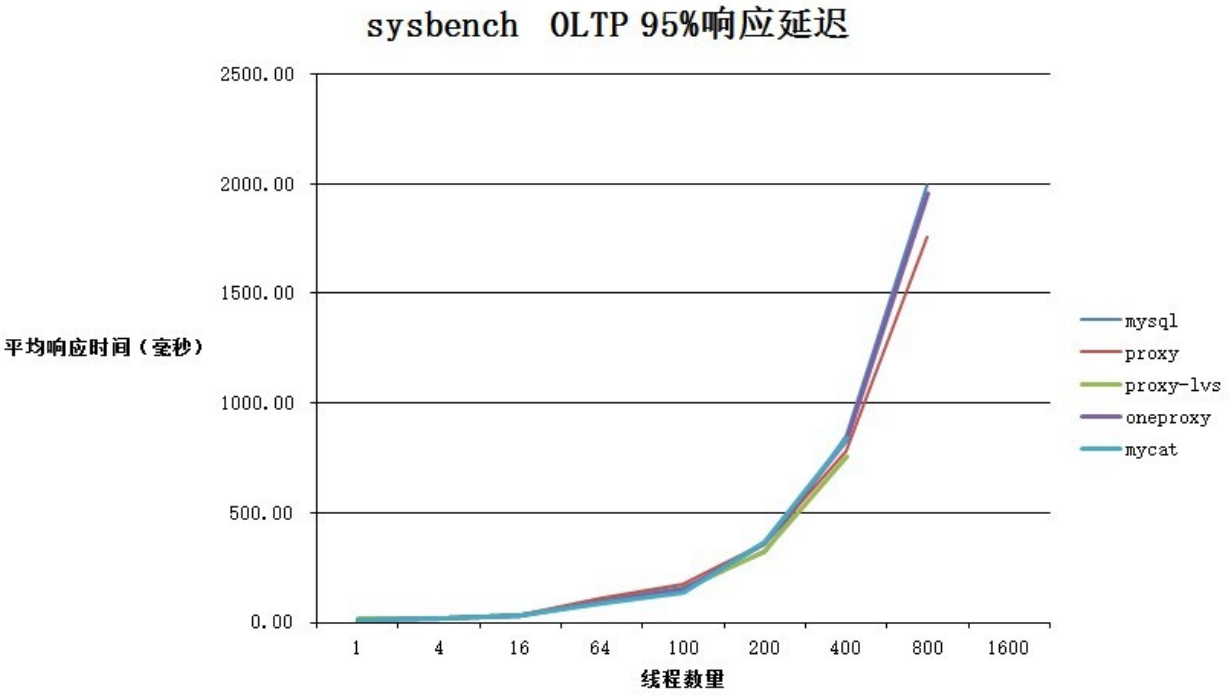
- QPS对比图



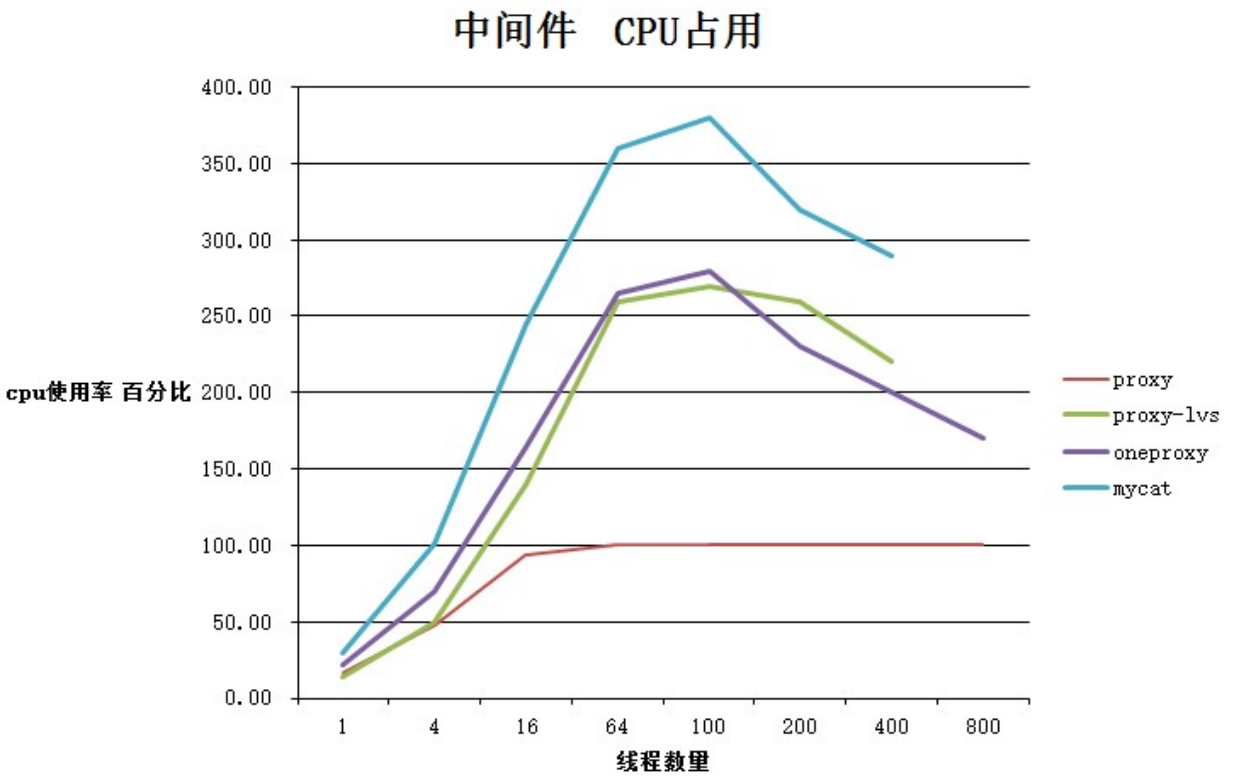
- 响应时间对比图



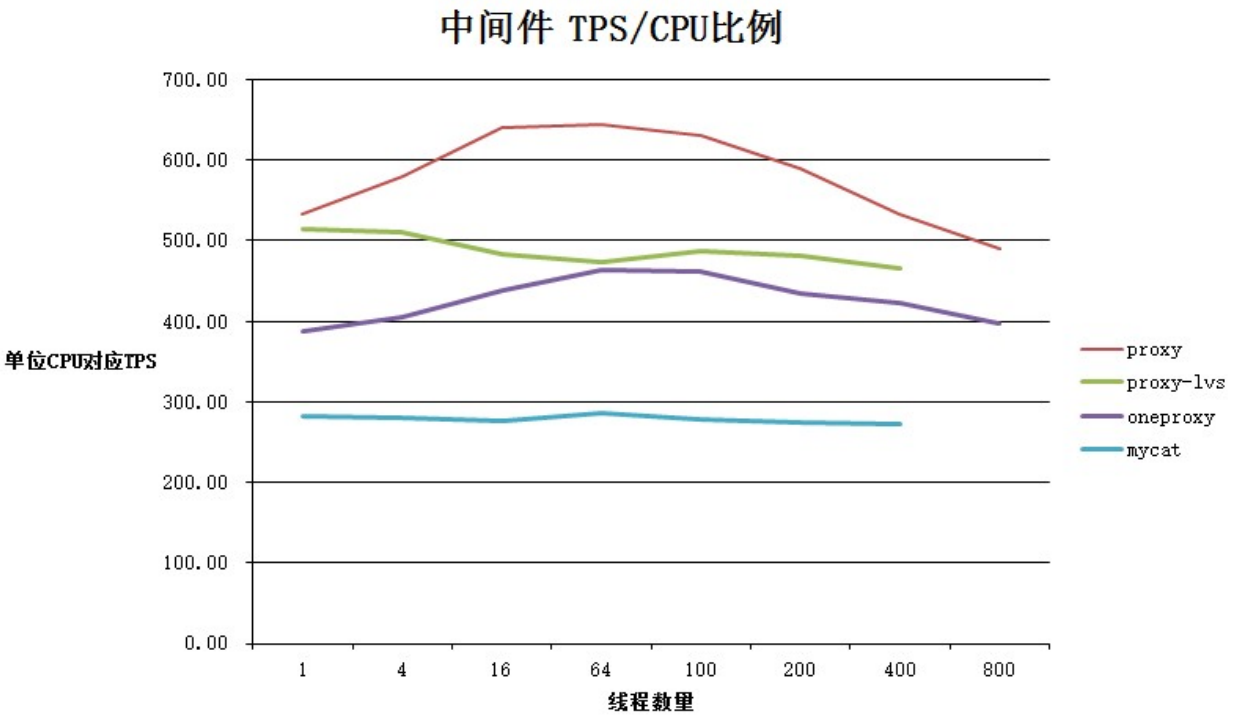
- 95%请求响应时间对比图



- cpu使用时间对比图



- TPS/CPU比例对比图



新机器10分钟事务测试

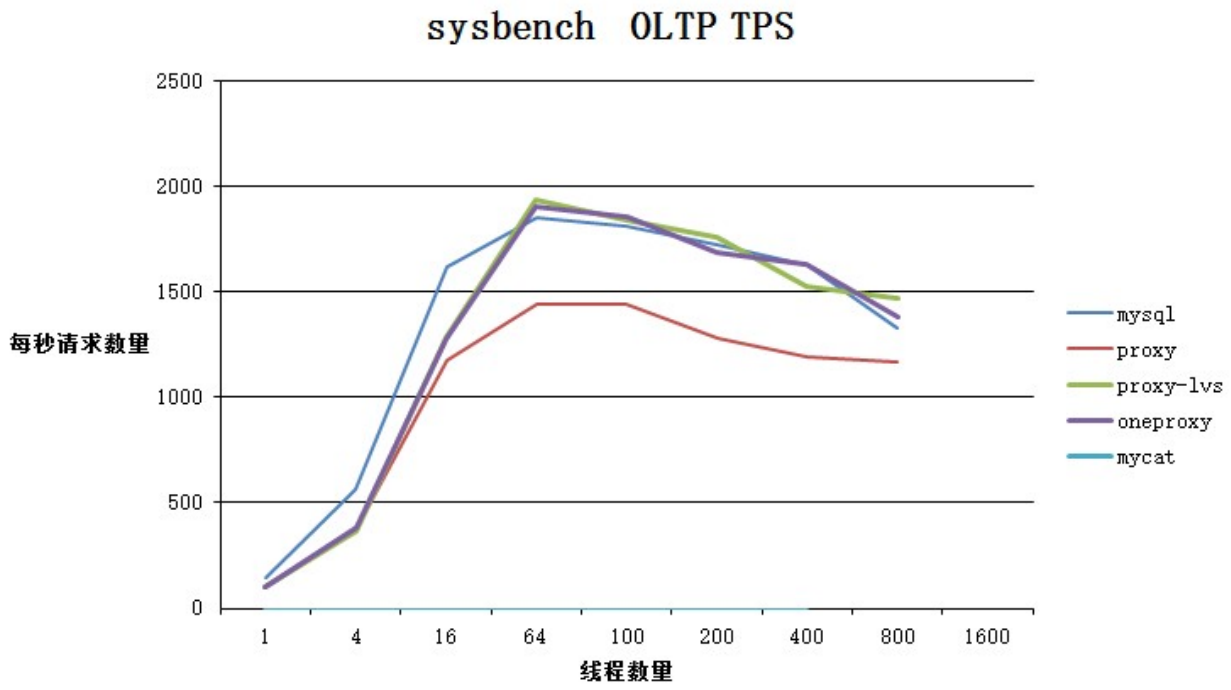
threads/MySQL	transaction/sec	request/sec.	reponse/ms	avg95/ms
1	145.39	2617.04	6.88	7.09
4	564.44	10159.94	7.09	7.80
16	1622.19	29199.38	9.86	11.24

64	1855.14	33392.65	34.50	39.61
100	1811.77	32612.03	55.19	68.11
200	1719.83	30957.14	116.29	148.46
400	1628.87	29320.16	245.56	354.86
800	1329.49	23931.36	601.71	938.49
threads/proxy	transaction/sec	request/sec.	reponse/ms	avg95/ms
1	106.73	1921.15	9.37	9.93
4	372.73	6709.19	10.73	11.95
16	1175.40	21157.16	13.61	14.84
64	1439.44	25909.91	44.46	46.10
100	1441.19	25941.38	69.38	71.03
200	1280.93	23056.97	156.13	171.71
400	1190.97	21437.70	335.85	387.74
800	1166.77	21002.48	685.61	846.41
threads/proxy+lvs	transaction/sec	request/sec.	reponse/ms	avg95/ms
1	97.31	1751.56	10.28	11.08
4	365.06	6571.06	10.96	11.98
16	1283.03	23094.62	12.47	13.62
64	1938.21	34887.80	33.02	39.62
100	1838.40	33091.33	54.39	65.22
200	1756.72	31621.02	113.85	136.28
400	1526.26	27473.16	262.07	397.61
800	1467.40	26414.01	545.16	862.78
threads/oneproxy	transaction/sec	request/sec.	reponse/ms	avg95/ms
1	102.88	1851.89	9.72	10.08
4	378.99	6821.76	10.55	11.60
16	1278.94	23020.99	12.51	13.91
64	1907.46	34334.34	33.55	38.10
100	1855.17	33393.17	53.90	64.60
200	1683.82	30308.75	118.77	141.27

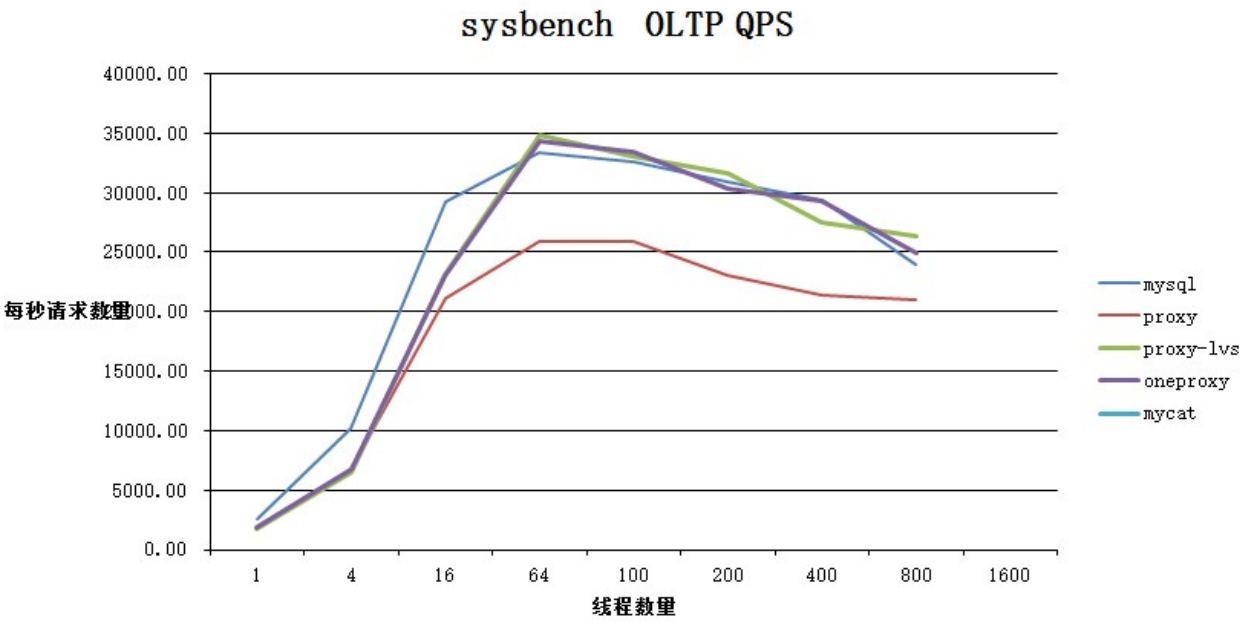
400	1626.44	29276.27	245.93	356.14
800	1384.76	24926.56	577.70	903.48

数据对比图

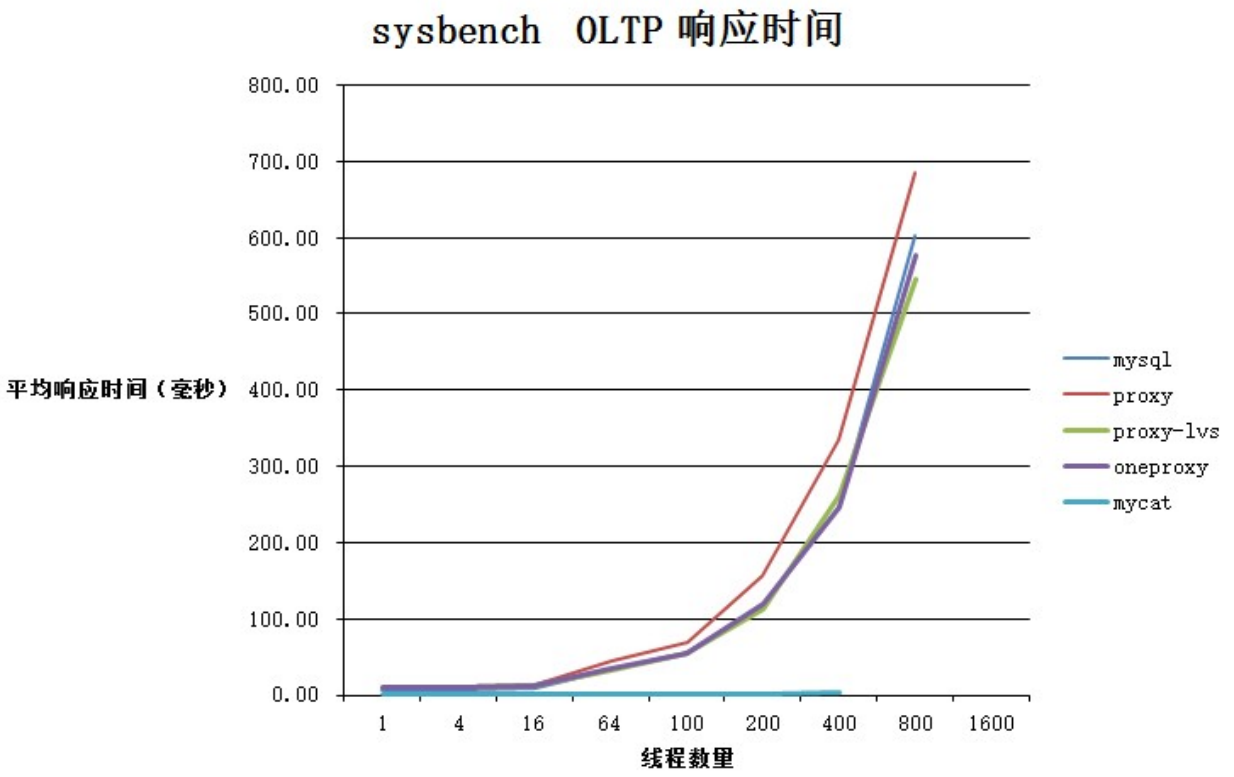
- tps对比图



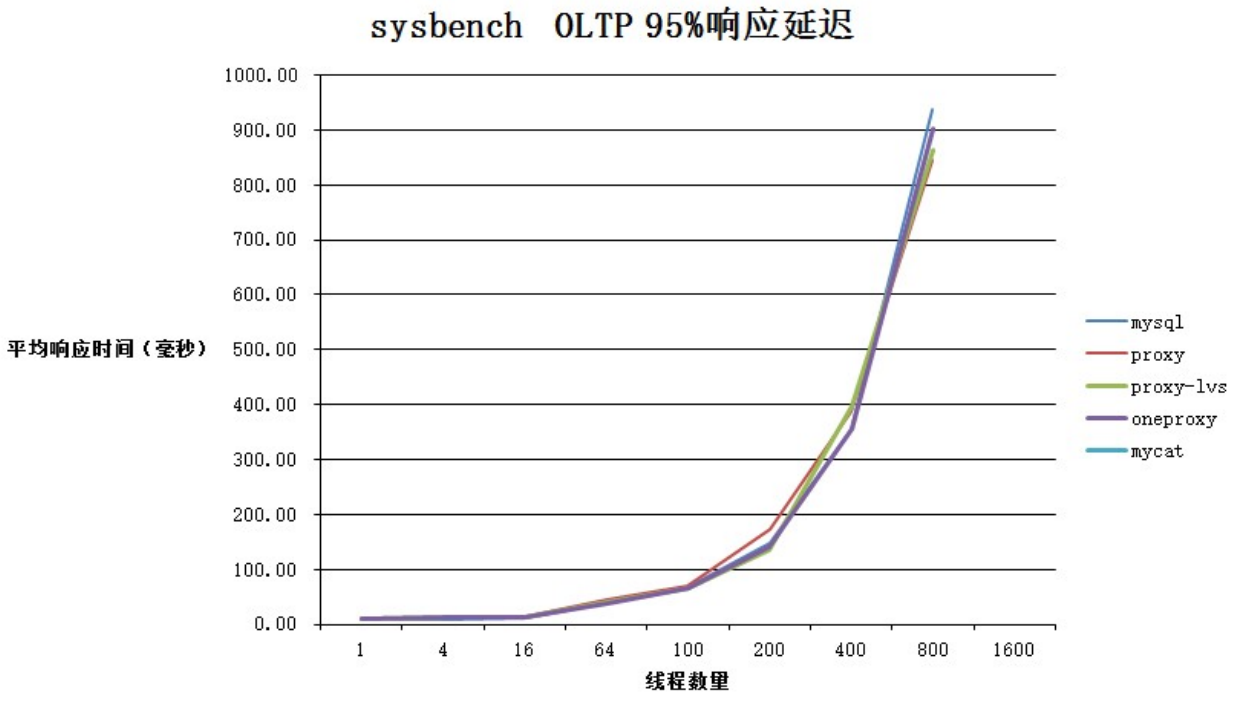
- QPS对比图



- 响应时间对比图



- 95%请求响应时间对比图



新机器10分钟读写分离测试

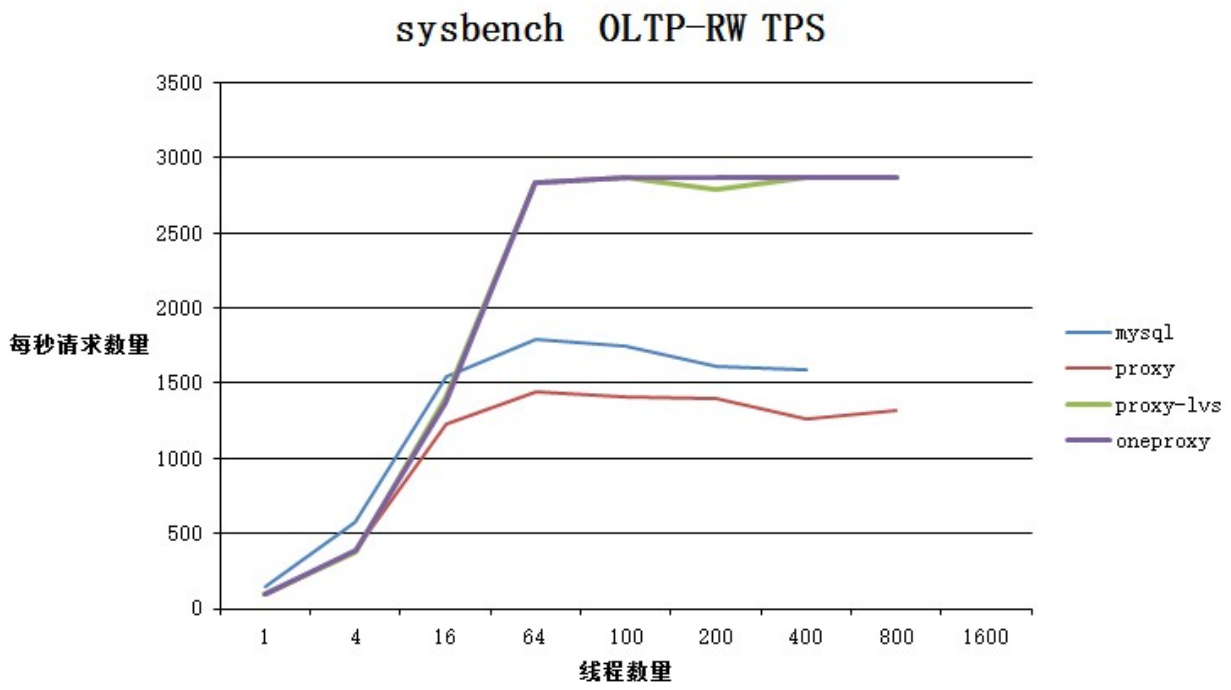
threads/MySQL	transaction/sec	request/sec.	reponse/ms	avg95/ms
1	151.68	2730.27	6.59	6.80
4	571.46	10286.28	7.00	7.49
16	1542.44	27763.93	10.37	11.74

64	1787.02	32166.52	35.81	40.98
100	1741.99	31355.84	57.40	70.01
200	1609.54	28971.72	124.26	167.55
400	1584.87	28527.99	252.38	363.35
threads/proxy	transaction/sec	request/sec.	reponse/ms	avg95/ms
1	88.96	1601.25	11.24	12.16
4	384.22	6915.97	10.41	11.10
16	1225.62	22061.13	13.05	14.23
64	1442.49	25964.80	44.37	45.62
100	1411.29	25403.31	70.85	72.58
200	1401.33	25224.16	142.71	155.19
400	1260.75	22693.97	317.26	351.69
800	1318.96	23741.92	606.30	642.66
threads/proxy+lvs	transaction/sec	request/sec.	reponse/ms	avg95/ms
1	93.32	1679.74	10.71	11.74
4	380.92	6856.64	10.50	11.42
16	1407.89	25342.04	11.36	12.41
64	2830.08	50941.51	22.61	26.95
100	2870.65	51671.69	34.83	46.94
200	2788.69	50196.78	71.71	109.34
400	2870.35	51666.68	139.25	170.33
800	2870.67	51672.65	278.50	313.03
threads/oneproxy	transaction/sec	request/sec.	reponse/ms	avg95/ms
1	91.51	1647.13	10.93	11.81
4	384.18	6915.25	10.41	11.14
16	1365.72	24582.95	11.71	12.72
64	2834.07	51013.38	22.58	30.16
100	2871.57	51688.34	34.82	43.21
200	2872.33	51701.91	69.62	84.49
400	2871.78	51692.57	139.27	161.98

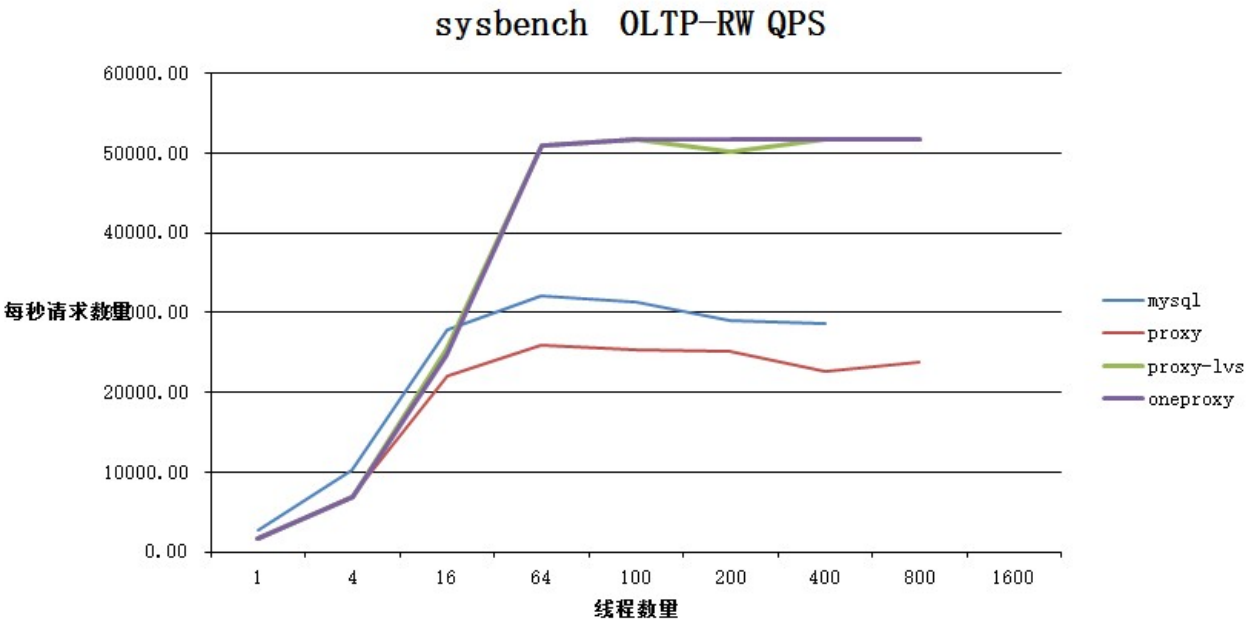
800	2868.09	51626.41	278.86	317.46

数据对比图

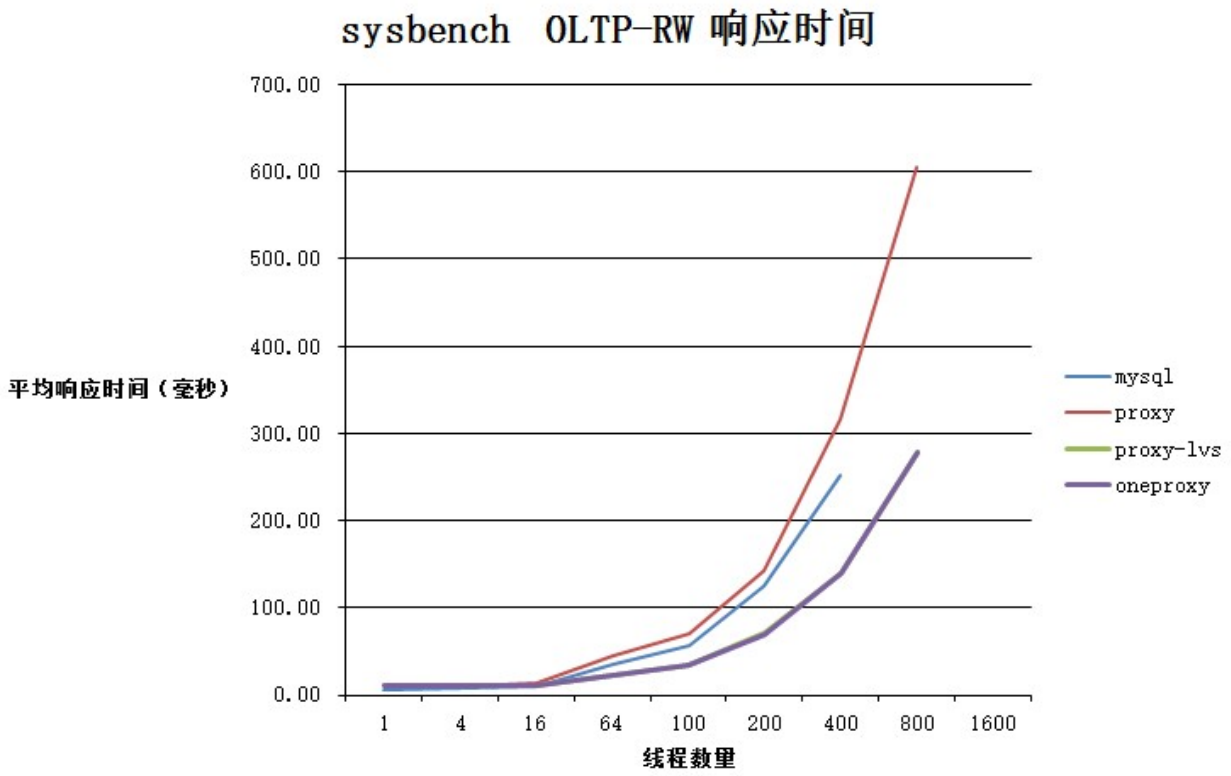
- tps对比图



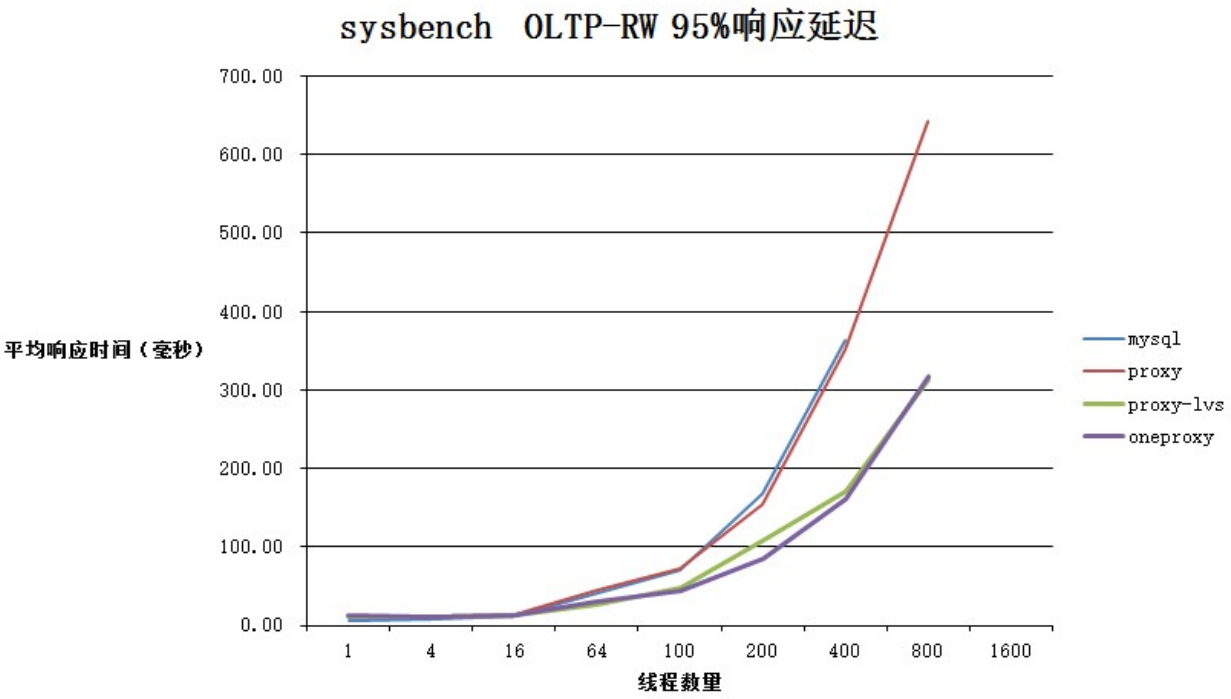
- QPS对比图



- 响应时间对比图



- 95%请求响应时间对比图



数据分析

单proxy性能分析

单机处理能力

在后端数据库未到达性能瓶颈之前，proxy性能随着并行执行线程数增长有下降，但是下降速率小于数据库本身处理能力下降速率。

在预期用于在线部署的机器上，在64连接以及以上的测试环境下。proxy能支持到1400左右的tps，对应25k的qps，基本能满足我们绝大部分的应用需求。

在在线测试环境下。由于数据库性能更好。proxy性能在64连接及之后，随着连接数的增加，性能基本没有出现衰减。在800连接并发测试情况下，仍然能达到1390的tps和25k的qps。基于event的连接处理模型保证了处理速度不会随着监听连接的增长而快速下降。但是因为cpu处理能力的限制，会给请求带来额外的延迟，延迟会随着连接数的增加而显著增大。

延迟分析

proxy会带来额外的延迟，额外延迟的表现体现在进行较少连接数测试时，proxy的tps和qps显著低于直连MySQL，在较少连接时，proxy可能会带来将近5ms的延迟。由于业务处理本身的延迟就小，所以在单线程密集业务处理的时候，性能比直连MySQL下降30%到40%。随着并发数的增加，由于proxy自身达到了cpu瓶颈，所带来的延迟也是增加的，平均响应延迟会比直连MySQL方式增加30%到40%。但是从95%响应时间来看，直连MySQL和proxy连接基本一致。说明整体延迟可控。

网络延迟

在新环境的测试中，网络带来的延迟稳定在0.2毫秒左右，不再可以忽略不计。和直接连接本机相比，一次完整的测试流程需要有19次和数据库的交互，带来的延迟就有3.8ms，而单进程直连MySQL的平均延迟只有8ms左右，网络带来的延迟就有将近一半。

多proxy集群部署

多proxy集群部署，采用lvs做前端。便于进行在线的业务维护。同时lvs也提供了故障迁移，负载均衡的功能，为软件的平稳运行提供了很好的条件。

多集群的proxy扩展性

在我们进行测试时，使用的是4个proxy并行提供服务。通过我们的测试延迟值对比，可以发现，lvs基本不会带来太多的额外延迟。

在处理量达到和超过单proxy处理阈值的时候，基于lvs方式部署的proxy集群可以很好的响应后端的需求。处理能力比直连MySQL稍差，但是增长曲线是一致的。

多集群的部署的proxy集群，由于未达到cpu性能瓶颈，不会因为proxy本身的处理能力限制带来额外的延迟，所以大连接数下的延迟的表现也和直连MySQL基本一致。在连接数比较少的情况下，因为proxy以及lvs的转发均会带来一些额外的延迟。所以，在连接数比较少的情况下，单连接性能比直连MySQL要低30%到40%。

在集群中的proxy达到或者接近cpu瓶颈之后，对应的数据访问延迟会有增加。DBA需要进行性能监控。在发现cpu使用较高时评估系统状态，决定是进行proxy的扩容还是保持现有状态不动。保护后端的数据库。

带宽瓶颈

在多线程并发的测试中，我们发现，在到达一定的tps之后。数据库压力不大，但是，性能数据上不去。后来观察发现，数据库本身是千兆网卡，而在tps达到2700+之后，响应的业务数据网卡带宽占用已经超过了90%，对于数据库主库而言，在响应数据的同时，还需要同步binlog给从库，在压力测试时，网卡带宽使用率一度接近100%，随着并发连接数的倍增，处理延迟基本翻倍。从数据分析看，网卡的瓶颈也是一个重点方向，当然，实际业务更加复杂，一般网卡都不会是瓶颈，但是，这块也是我们后续需要重点关注的点，如果允许的话，最好数据库也使用bonding接入网络，减少硬件故障导致问题的同时，也能在平时提升可用带宽。

proxy需要做一次数据的转发工作，所有从server收到的数据，都需要再发送到client端。由于proxy的内网做了bonding，可以缓解一部分这个问题，测试时，proxy服务器内网的流量一度超过1.9Gbps。

数据库性能瓶颈

在测试环境下，从直连MySQL和lvs+proxy的数据来看，64线程并发测试情况下，基本就已经到达了数据库的性能瓶颈。数据库到达瓶颈之后。直连MySQL和lvs+proxy的性能基本一致。但是根据实际的观察，网络方面的瓶颈应该也是性能上不去的原因之一。后续采用了读写分离之后，性能还是有一个相对的瓶颈。这个瓶颈应该也是网络方面的限制导致的。

在未采用读写分离的情况下，所有的读写压力都由数据库承担，在并行执行的线程数超过了MySQL所在服务器的cpu的核数的两倍之后，性能开始恶化，同时，服务器的负载开始飙升，在800线程测试时，服务器的负载能达到200+。

proxy与友商软件的对比

我们简单对比了下oneproxy和mycat，整体表现的话，oneproxy部署便利性和性能相对来说是最好的。性能表现大部分情况下和proxy+lvs相当，在读写分离的情况下，还比proxy+lvs 稍占优势。同时，他支持的读写分离策略也比较多。但是，oneproxy的使用硬限制比较多。不支持prepare_statement导致tpcc的测试无法进行。每个用户必须指定一个特定的数据库这点也和我们的使用习惯不同。

mycat的性能表现也不错。单节点部署就可以很好的利用cpu资源，但是配置起来相对麻烦，所有的表和库的映射都需要手动配置到mycat中，一旦发生变化，需要停机维护进行处理。同时，和C程序相比，cpu使用效率相对低一点。而且mycat对于事务的处理和部分错误的处理不太友好，比如在新机器上，因为数据库处理并行化程度增加，在测试中常有事务死锁需要重试的情况出现，这种情况下mycat会认为连接异常直接关闭连接，导致测试软件没法继续测试，所以新环境下的测试数据都没有mycat相关的数据。

读写分离的表现

读写分离是直观的快速扩展MySQL业务能力的方式。通过测试，我们可以看到。分离部分不需要强一致性的读请求到从节点上，减轻主服务器的压力，能快速的提升业务处理能力。在测试这种读多写少的情况下，一个读服务器就能提升将近一倍的处理性能。并且，性能在达到网卡性能瓶颈后能较长时间都保持稳定。当然，实际的业务场景远没有测试场景理想，在实际业务的处理情况下，业务往往更加复杂，对MySQL的cpu资源依赖更高。这种情况下的读写分离，也能有效的降低主服务器的cpu资源开销。有效的保证业务正常进行。

tpcc测试数据

测试数据介绍

测试场景

根据业务的需求，我们选取了业界相对比较认可的tpcc测试对系统进行了一轮测试。

由于是针对MySQL库进行的，所以我们选取了tpcc-mysql测试套件进行了一轮性能测试。

测试维度

本次性能测试主要是横向对比，对比直连MySQL和我们在业务部署场景下的性能对比。得出proxy的部署在业务情况下会带来的性能开销，为后续我们系统上线做相关的准备。

由于tpcc的测试语句都在事务中，且时间关系，没法修改tpcc代码实现读写分离。故本次测试没有测试读写分离环境下的tpcc性能表现，后续有时间，可以考虑对代码进行进一步的修改，以验证功能的提升。

测试软件配置

我们使用了100个warehouse的配置模式。进行相关的测试。测试前，对数据进行了充分的预热，保证系统和数据库都是工作在较好的情况下。同时，针对之前的情况，我们发现100个线程的情况下，MySQL数据库有较好的性能表现。所以，我们本次测试使用了并发100个线程进行测试。

测试数据导入后，已经进行了一个小时的相关验证测试，同时进行数据库的预热。

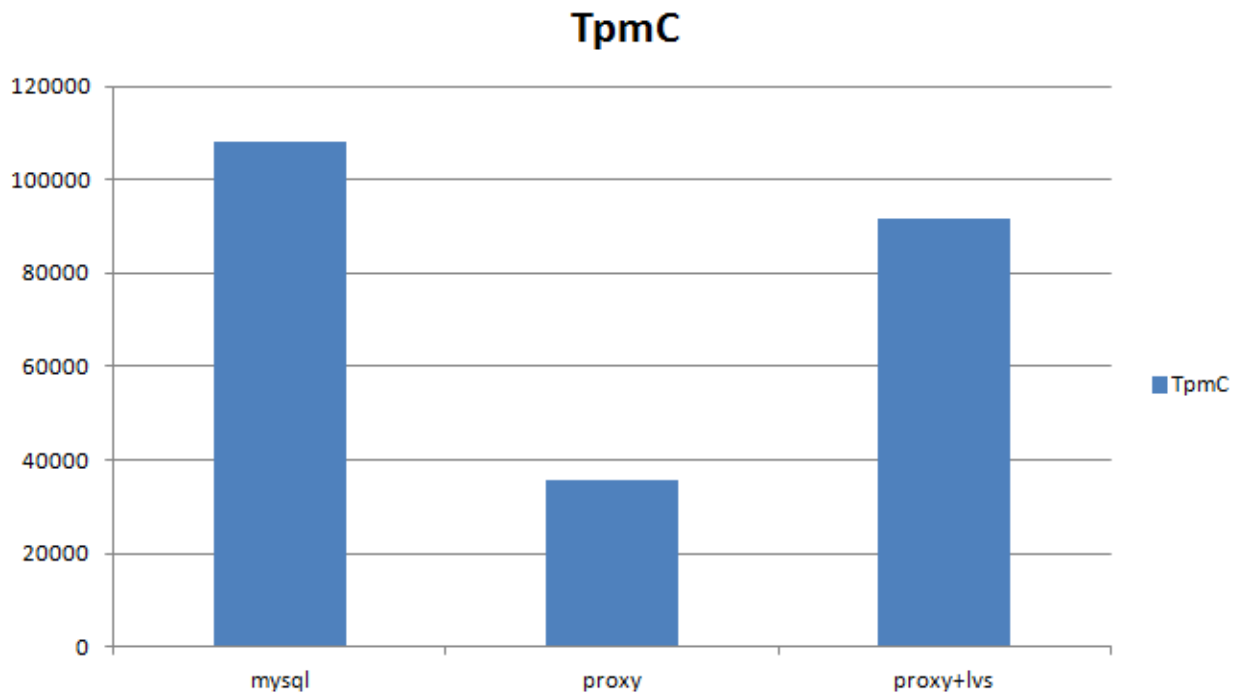
每次测试前5分钟是预热，之后运行1小时，得到相关处理数据。

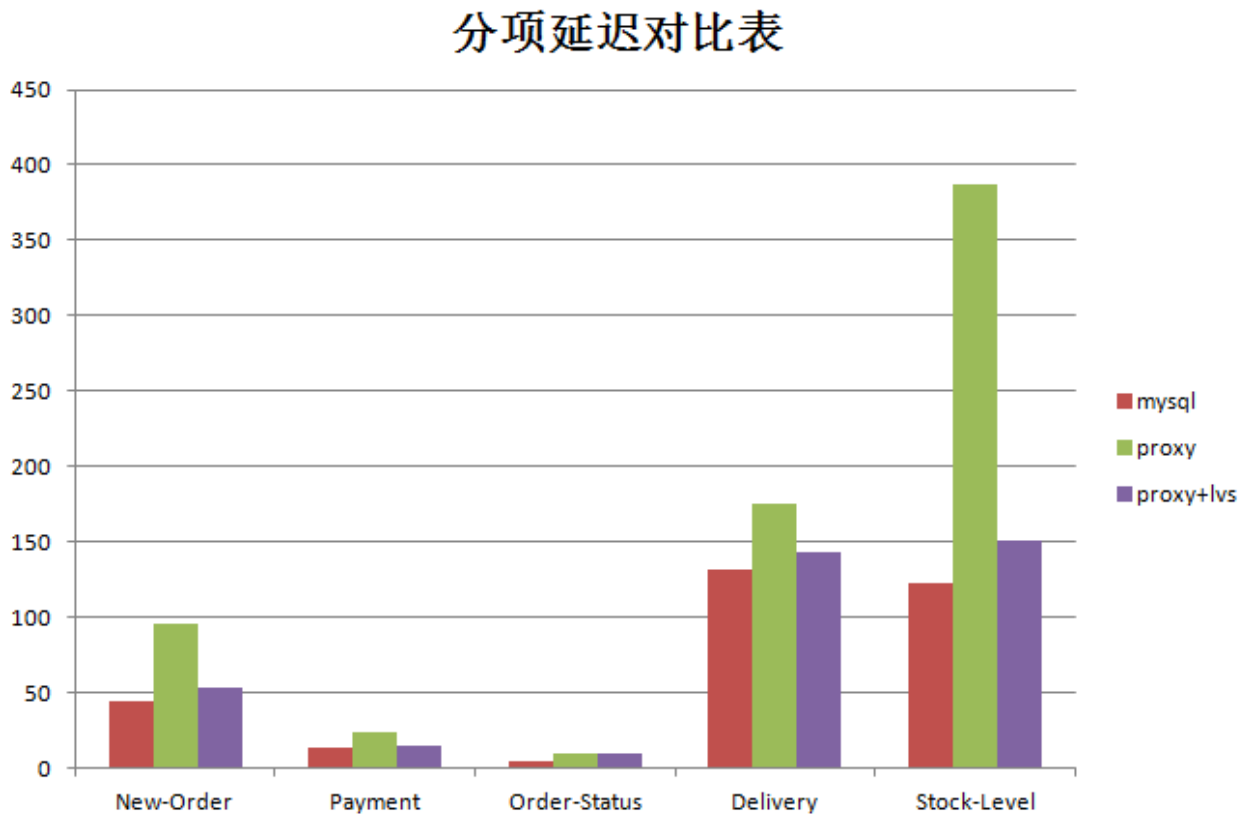
测试用的初始化命令行和相关的测试时执行的脚本可以参看附录中提供的文件。

测试数据

TPCC	TpmC	New-Order(ms)	Payment(ms)	Order-Status(ms)	Delivery(ms)
MySQL	108242.8	44	13.2	4.7	131.8
proxy	35611.301	95.3	24.3	9.7	175.4
proxy+lvs	91519.797	53.2	14.7	9.5	143.1

数据对比图





数据分析

在测试时，观察了下MySQL数据库的情况，数据库的cpu使用率均在2800%以上。cpu使用率较高。

tpcc的测试和网络瓶颈无关，因为tpcc测试时，网络带宽只使用了500Mbps左右。只有带宽上限的50%，有较大的余量。

单就处理能力来看，直连MySQL，以及proxy+lvs的方式，对数据的处理能力和延迟都没有太大的区别。

proxy+lvs方式性能比直连MySQL要低10%左右，是因为在某些情况下，4个proxy仍会有cpu跑满的情况，影响了系统的发挥，同时，proxy带来的延迟也会相应的对整体的处理性能有一定的影响。

单个proxy对tpcc的测试是有很大影响的，具体影响的原因主要也是因为单进程工作模式带来的cpu等待。当然，单个proxy就能达到的35000的TpmC也是一个相对比较高的数值了。应付一般的业务都是足够的。

附录：

软件的配置信息

MySQL(master)

```
[client]
port            = 3306
socket          = /tmp/mysql_3306.sock

# The MySQL server
[mysqld]
open_files_limit = 65535
port            = 3306
socket          = /tmp/mysql_3306.sock
```

```
performance_schema=ON
skip_name_resolve=ON
character-set-server=utf8
basedir=/home/mysql3306/mysqlhome
tmpdir=/home/mysql3306/mysqlhome/tmp
innodb_data_home_dir=/ssddata1/mysql3306/data
datadir = /ssddata1/mysql3306/data
#slave_skip_errors=all
#slave_skip_errors=1062
master-info-repository=TABLE
relay-log-info-repository=TABLE
sync-master-info=1
sync_binlog=10000
slow_query_log
long_query_time = 1
skip_name_resolve
skip-external-locking
expire_logs_days=7
max_binlog_size=1G
back_log = 2000
max_connections = 10000
lower_case_table_names = 1
max_connect_errors = 1000
table_open_cache = 4096
key_buffer_size = 128M
max_allowed_packet = 16M
binlog_cache_size = 1M
max_heap_table_size = 64M
sort_buffer_size = 2M
read_buffer_size = 2M
join_buffer_size=4M
read_rnd_buffer_size = 8M
myisam_sort_buffer_size = 64M
thread_cache_size = 64
query_cache_size = 0
query_cache_type = 0
query_cache_limit = 0
default-storage-engine = InnoDB
thread_stack = 192K
transaction_isolation = READ-COMMITTED
tmp_table_size = 64M
log-bin=mysql-bin
log_slave_updates=on
binlog_format=ROW
key_buffer_size = 32M
bulk_insert_buffer_size = 64M
myisam_sort_buffer_size = 128M
myisam_max_sort_file_size = 5G
myisam_repair_threads = 1
#myisam_recover
#innodb_additional_mem_pool_size = 16M
innodb_data_file_path = ibdata1:10M:autoextend
```

```
innodb_write_io_threads = 8
innodb_read_io_threads = 8
innodb_thread_concurrency = 0
innodb_flush_log_at_trx_commit = 0
innodb_log_buffer_size = 16M
innodb_log_file_size = 512M
innodb_log_files_in_group = 3
innodb_max_dirty_pages_pct = 50
innodb_max_dirty_pages_pct_lwm=20
innodb_lock_wait_timeout = 120
innodb_file_per_table=on
innodb_online_alter_log_max_size=512M
log-error
innodb_buffer_pool_size=40G
server-id = 23088
#thread_concurrency = 40
character-set-server = utf8
max_allowed_packet = 32M
thread_cache_size = 32
log_bin_trust_function_creators=1
innodb_purge_threads=2
innodb_flush_neighbors=0
####Percentage of log capacity below which no adaptive flushing happens.
innodb_adaptive_flushing_lwm=10
innodb_io_capacity=2000
innodb_io_capacity_max=4000
table_open_cache_instances=16
innodb_flush_method=O_DIRECT
table_open_cache_instances=16
gtid_mode=on
enforce_gtid_consistency=on

[mysqldump]
socket          = /tmp/mysql_3306.sock
quick
max_allowed_packet = 16M
default-character-set=utf8
[mysql]
socket          = /tmp/mysql_3306.sock
no-auto-rehash
[myisamchk]
key_buffer_size = 256M
sort_buffer_size = 256M
read_buffer = 2M
write_buffer = 2M

[mysqlhotcopy]
interactive-timeout
```

MySQL(slave)

```
```

```
[client]
port            = 3307
socket          = /tmp/mysql_3307.sock

# The MySQL server
[mysqld]
open_files_limit = 65535
port            = 3307
socket          = /tmp/mysql_3307.sock
performance_schema=ON
skip_name_resolve=ON
character-set-server=utf8
basedir=/home/mysql3307/mysqlhome
tmpdir=/home/mysql3307/mysqlhome/tmp
innodb_data_home_dir=/ssddata1/mysql3307/data
datadir = /ssddata1/mysql3307/data
#slave_skip_errors=all
#slave_skip_errors=1062
master-info-repository=TABLE
relay-log-info-repository=TABLE
sync-master-info=1
sync_binlog=10000
slow_query_log
long_query_time = 1
skip_name_resolve
skip-external-locking
expire_logs_days=7
max_binlog_size=1G
back_log = 50
max_connections = 10000
lower_case_table_names = 1
max_connect_errors = 1000
table_open_cache = 4096
key_buffer_size = 128M
max_allowed_packet = 16M
binlog_cache_size = 1M
max_heap_table_size = 64M
sort_buffer_size = 2M
read_buffer_size = 2M
join_buffer_size=4M
read_rnd_buffer_size = 8M
myisam_sort_buffer_size = 64M
thread_cache_size = 64
query_cache_size = 0
query_cache_type = 0
query_cache_limit = 0
default-storage-engine = InnoDB
thread_stack = 192K
transaction_isolation = READ-COMMITTED
tmp_table_size = 64M
log-bin=mysql-bin
log_slave_updates=on
```

```
binlog_format=ROW
key_buffer_size = 32M
bulk_insert_buffer_size = 64M
myisam_sort_buffer_size = 128M
myisam_max_sort_file_size = 5G
myisam_repair_threads = 1
#myisam_recover
#innodb_additional_mem_pool_size = 16M
innodb_data_file_path = ibdata1:10M:autoextend
innodb_write_io_threads = 8
innodb_read_io_threads = 8
innodb_thread_concurrency = 0
innodb_flush_log_at_trx_commit = 0
innodb_log_buffer_size = 16M
innodb_log_file_size = 512M
innodb_log_files_in_group = 3
innodb_max_dirty_pages_pct = 50
innodb_max_dirty_pages_pct_lwm=20
innodb_lock_wait_timeout = 120
innodb_file_per_table=on
innodb_online_alter_log_max_size=512M
log-error
innodb_buffer_pool_size=50G
server-id = 288
#thread_concurrency = 40
character-set-server = utf8
max_allowed_packet = 32M
thread_cache_size = 32
log_bin_trust_function_creators=1
innodb_purge_threads=2
innodb_flush_neighbors=0
####Percentage of log capacity below which no adaptive flushing happens.
innodb_adaptive_flushing_lwm=10
innodb_io_capacity=2000
innodb_io_capacity_max=4000
table_open_cache_instances=16
innodb_flush_method=O_DIRECT
table_open_cache_instances=16
gtid_mode=on
enforce_gtid_consistency=on

[mysqldump]
socket          = /tmp/mysql_3307.sock
quick
max_allowed_packet = 16M
default-character-set=utf8
[mysql]
socket          = /tmp/mysql_3307.sock
no-auto-rehash
[myisamchk]
key_buffer_size = 256M
sort_buffer_size = 256M
```

```
read_buffer = 2M
write_buffer = 2M

[mysqlhotcopy]
interactive-timeout
```

proxy(final-single)

```
[mysql-proxy]
daemon = true
pid-file = /home/mysql-cetus/cetus_install/bin/mysql-proxy.pid
log-file = /home/mysql-cetus/cetus_install/bin/mysql-proxy.log
log-level = debug
max-open-files = 10240
plugins = admin,proxy
plugin-dir=/home/mysql-cetus/cetus_install/lib/mysql-proxy/plugins
proxy-address = 0.0.0.0:3307
proxy-backend-addresses = host4:3306
proxy-read-only-backend-addresses = host4:3307
admin-address = 127.0.0.1:4041
admin-lua-script = /home/mysql-cetus/cetus_install/lib/mysql-proxy/lua/admin.lua
admin-username = admin
admin-password = admin_pass


disable-sharding-mode=true

default-username=test
default-db=
default-pool-size=500
user-pwd=test@xxxxxx
app-user-pwd=test@xxxxxx
```

MySQLproxy(final-lvs)

proxy的4个进程每个进程的启动都有点小的不同。我们以5440的配置文件为例，他的对应修改下文件名和监听的端口即可。

```
[mysql-proxy]
daemon = true
pid-file = /home/mysql-cetus/cetus_install/lvs_run/my_5440.pid
log-file = /home/mysql-cetus/cetus_install/lvs_run/my_5440.log
log-level = debug
max-open-files = 10240
plugins = admin,proxy
plugin-dir=/home/mysql-cetus/cetus_install/lib/mysql-proxy/plugins
proxy-address = 0.0.0.0:5440
proxy-backend-addresses = host4:3306
proxy-read-only-backend-addresses = host4:3307
admin-address = 127.0.0.1:5441
admin-lua-script = /home/mysql-cetus/cetus_install/lib/mysql-proxy/lua/admin.lua
admin-username = admin
admin-password = admin_pass


disable-sharding-mode=true

default-username=test
default-db=
default-pool-size=250
user-pwd=test@xxxxxx
app-user-pwd=test@xxxxxx
```

keepalived配置

我们需要keepalived进行配置。保证我们可以通过统一的端口访问后面的4个服务。

```
! Configuration File for keepalived

global_defs {
   notification_email {
     weihe@corp.netease.com
   }
   notification_email_from keepalived@host2_1
   smtp_server 127.0.0.1
   smtp_connect_timeout 30
   router_id rd8
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 8
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111243
    }
```

```
    virtual_ipaddress {
    }
}

virtual_server host2_2 3440 {
    delay_loop 6
    lb_algo wlc
    lb_kind FNAT
    persistence_timeout 0
    protocol TCP

    real_server host3_1 5440 {
        weight 10
        TCP_CHECK {
            connect_ip host3_1
            connect_port 5440
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
        }
    }

    real_server host3_1 5442 {
        weight 10
        TCP_CHECK {
            connect_ip host3_1
            connect_port 5442
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
        }
    }

    real_server host3_1 5444 {
        weight 10
        TCP_CHECK {
            connect_ip host3_1
            connect_port 5444
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
        }
    }

    real_server host3_1 5446 {
        weight 10
        TCP_CHECK {
            connect_ip host3_1
            connect_port 5446
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
```

```
        }
    }
}
```

测试环境测试脚本

```
#!/bin/bash
## Prepare...
MAXTIME=600
INTERVAL=10
SLEEPTIME=120
export LANG=C
#sysbench --test=/home/mysql-cetus/sysbench/sysbench/tests/db/oltp.lua --oltp-table-size=100000 --mysql-table-engine=innodb --mysql-user=test --mysql-password=xxxxxx --mysql-port=3306 --mysql-host=host4 --mysql-db=oltp --max-requests=0 --max-time=180 --oltp-tables-count=10 --report-interval=$INTERVAL --num-threads=50 prepare
for threads in 1 4 16 64 100 200 400 800 1600 ;do
echo mysql
date +"%Y-%m-%d %H:%M:%S"
date +%s
sysbench --test=/home/mysql-cetus/sysbench/sysbench/tests/db/oltp.lua --oltp-table-size=1000000 --mysql-table-engine=innodb --mysql-user=test --mysql-password=xxxxxx --max-requests=0 --max-time=$MAXTIME --oltp-tables-count=1 --report-interval=$INTERVAL --num-threads=$threads --mysql-port=3306 --mysql-host=host4 --mysql-db=oltp run
date +"%Y-%m-%d %H:%M:%S"
date +%s
sleep $SLEEPTIME
echo final
date +"%Y-%m-%d %H:%M:%S"
date +%s
sysbench --test=/home/mysql-cetus/sysbench/sysbench/tests/db/oltp.lua --oltp-table-size=1000000 --mysql-table-engine=innodb --mysql-user=test --mysql-password=xxxxxx --max-requests=0 --max-time=$MAXTIME --oltp-tables-count=1 --report-interval=$INTERVAL --num-threads=$threads --mysql-port=3307 --mysql-host=host3_1 --mysql-db=oltp run
date +"%Y-%m-%d %H:%M:%S"
date +%s
sleep $SLEEPTIME
date +"%Y-%m-%d %H:%M:%S"
date +%s
echo final+lvs
sysbench --test=/home/mysql-cetus/sysbench/sysbench/tests/db/oltp.lua --oltp-table-size=1000000 --mysql-table-engine=innodb --mysql-user=test --mysql-password=xxxxxx --max-requests=0 --max-time=$MAXTIME --oltp-tables-count=1 --report-interval=$INTERVAL --num-threads=$threads --mysql-port=3440 --mysql-host=host3_2 --mysql-db=oltp run
date +"%Y-%m-%d %H:%M:%S"
date +%s
sleep $SLEEPTIME
done
```

在线环境测试脚本

由于在线环境上，数据库和proxy未在同一个交换机上，且sysbench对测试要求较高。必须所有连接都成功建立并执行成功，测试才能继续，部分测试有失败的情况，所以在线的测试脚本增加了重试，我们只取能正确执行完的结果数据。

```
#!/bin/bash
## Prepare...
MAXTIME=300
INTERVAL=5
SLEEPTIME=60
USLEEP_INTERVAL=0
export LANG=C

if [ "x$1" == "xrw" ]; then
  LUASCRIPT=oltp.weihe.lua
else
  LUASCRIPT=oltp.lua
fi

print_date(){
  date +"%Y-%m-%d %H:%M:%S"
  date +%s
}

exec_bench() {
  echo $1 $2
  threads=$2
  case $1 in
    "mysql")
      CMD="sysbench --usleep-interval=$USLEEP_INTERVAL --forced-shutdown --
test=/home/mysql-cetus/sysbench/sysbench/tests/db/$LUASCRIPT --oltp-table-
size=100000 --mysql-table-engine=innodb --mysql-user=test --mysql-password=xxxxxx
--max-requests=0 --max-time=$MAXTIME --oltp-tables-count=10 --report-
interval=$INTERVAL --num-threads=$threads --mysql-port=3306 --mysql-host=host8_1
--mysql-db=oltp run"
    ;;
    "final")
      CMD="sysbench --usleep-interval=$USLEEP_INTERVAL  --forced-shutdown --
test=/home/mysql-cetus/sysbench/sysbench/tests/db/$LUASCRIPT --oltp-table-
size=100000 --mysql-table-engine=innodb --mysql-user=test --mysql-password=xxxxxx
--max-requests=0 --max-time=$MAXTIME --oltp-tables-count=10 --report-
interval=$INTERVAL --num-threads=$threads --mysql-port=3307 --mysql-host=host6 --
mysql-db=oltp run"
    ;;
    "final+lvs")
      CMD="sysbench --usleep-interval=$USLEEP_INTERVAL  --forced-shutdown --
test=/home/mysql-cetus/sysbench/sysbench/tests/db/$LUASCRIPT --oltp-table-
size=100000 --mysql-table-engine=innodb --mysql-user=test --mysql-password=xxxxxx
--max-requests=0 --max-time=$MAXTIME --oltp-tables-count=10 --report-
interval=$INTERVAL --num-threads=$threads --mysql-port=3440 --mysql-host=host5 --
mysql-db=oltp run"
    ;;
    "oneproxy")
```

```
      CMD="sysbench --usleep-interval=$USLEEP_INTERVAL  --forced-shutdown --
test=/home/mysql-cetus/sysbench/sysbench/tests/db/$LUASCRIPT --oltp-table-
size=100000 --mysql-table-engine=innodb --mysql-user=test --mysql-password=xxxxxx
--max-requests=0 --max-time=$MAXTIME --oltp-tables-count=10 --report-
interval=$INTERVAL --num-threads=$threads --mysql-port=3310 --mysql-host=host6 --
mysql-db=oltp run"
    ;;
    "mycat")
      CMD="sysbench --usleep-interval=$USLEEP_INTERVAL  --forced-shutdown --
test=/home/mysql-cetus/sysbench/sysbench/tests/db/$LUASCRIPT --oltp-table-
size=100000 --mysql-table-engine=innodb --mysql-user=test --mysql-password=xxxxxx
--max-requests=0 --max-time=$MAXTIME --oltp-tables-count=10 --report-
interval=$INTERVAL --num-threads=$threads --mysql-port=8066 --mysql-host=host6 --
mysql-db=oltp run"
    ;;
  esac
  retry=0
  echo $CMD
  print_date
  eval $CMD
  while [ $? -ne 0 ] && [ $retry -lt 3 ]; do
    sleep 10;
    echo try $1 agine..
    retry=$((retry+1))
    print_date
    eval $CMD
  done
  print_date
  if [ $2 -le 16 ]; then
    sleep 10;
  else
    sleep $SLEEPTIME
  fi
}

#sysbench --test=/home/mysql-cetus/sysbench/sysbench/tests/db/oltp.lua --oltp-
table-size=100000 --mysql-table-engine=innodb --mysql-user=test --mysql-
password=xxxxxx --mysql-port=3306 --mysql-host=host8_1 --mysql-db=oltp --max-
requests=0 --max-time=180 --oltp-tables-count=10 --report-interval=$INTERVAL --
num-threads=50 prepare
for threads in 1 4 16 64 100 200 400 800 1600 ;do
  for software in mysql final final+lvs oneproxy mycat ; do
    exec_bench $software $threads
  done
done
```

读写分离测试用lua

修改后的sysbench用于读写分离的lua脚本如下：

```
pathtest = string.match(test, "(.*/)") or ""
```

```
dofile(pathtest .. "common.lua")

function thread_init(thread_id)
   set_vars()

   if (((db_driver == "mysql") or (db_driver == "attachsql")) and
mysql_table_engine == "myisam") then
      begin_query = "LOCK TABLES sbtest WRITE"
      commit_query = "UNLOCK TABLES"
   else
      begin_query = "BEGIN"
      commit_query = "COMMIT"
   end

end

function event(thread_id)
   local rs
   local i
   local table_name
   local range_start
   local c_val
   local pad_val
   local query

   table_name = "sbtest".. sb_rand_uniform(1, oltp_tables_count)

   for i=1, oltp_point_selects do
      rs = db_query("SELECT c FROM ".. table_name .." WHERE id=" .. sb_rand(1,
oltp_table_size))
   end

   for i=1, oltp_simple_ranges do
      range_start = sb_rand(1, oltp_table_size)
      rs = db_query("SELECT c FROM ".. table_name .." WHERE id BETWEEN " ..
range_start .. " AND " .. range_start .. "+" .. oltp_range_size - 1)
   end

   for i=1, oltp_sum_ranges do
      range_start = sb_rand(1, oltp_table_size)
      rs = db_query("SELECT SUM(K) FROM ".. table_name .." WHERE id BETWEEN " ..
range_start .. " AND " .. range_start .. "+" .. oltp_range_size - 1)
   end

   for i=1, oltp_order_ranges do
      range_start = sb_rand(1, oltp_table_size)
      rs = db_query("SELECT c FROM ".. table_name .." WHERE id BETWEEN " ..
range_start .. " AND " .. range_start .. "+" .. oltp_range_size - 1 .. " ORDER BY
c")
   end
```

```
   if not oltp_skip_trx then
      db_query(begin_query)
   end

   for i=1, oltp_distinct_ranges do
      range_start = sb_rand(1, oltp_table_size)
      rs = db_query("SELECT DISTINCT c FROM ".. table_name .." WHERE id BETWEEN "
.. range_start .. " AND " .. range_start .. "+" .. oltp_range_size - 1 .. " ORDER
BY c")
   end

   if not oltp_read_only then

   for i=1, oltp_index_updates do
      rs = db_query("UPDATE " .. table_name .. " SET k=k+1 WHERE id=" ..
sb_rand(1, oltp_table_size))
   end

   for i=1, oltp_non_index_updates do
      c_val = sb_rand_str("###########-###########-###########-###########-
###########-###########-###########-###########-###########-###########")
      query = "UPDATE " .. table_name .. " SET c='" .. c_val .. "' WHERE id=" ..
sb_rand(1, oltp_table_size)
      rs = db_query(query)
      if rs then
        print(query)
      end
   end

   i = sb_rand(1, oltp_table_size)

   rs = db_query("DELETE FROM " .. table_name .. " WHERE id=" .. i)

   c_val = sb_rand_str([[
###########-###########-###########-###########-###########-###########-
###########-###########-###########-###########]])
   pad_val = sb_rand_str([[
###########-###########-###########-###########-###########]])

   rs = db_query("INSERT INTO " .. table_name ..  " (id, k, c, pad) VALUES " ..
string.format("(%d, %d, '%s', '%s')",i, sb_rand(1, oltp_table_size) , c_val,
pad_val))

   end -- oltp_read_only

   if not oltp_skip_trx then
      db_query(commit_query)
   end

end
```

tpcc并行初始化脚本load_warehouse_para.sh

```
export LD_LIBRARY_PATH=/usr/local/mysql/lib/mysql/
DBNAME=$1
WH=$2
HOST=host
if [ "x$3" == "x" ]; then
STEP=10
else
STEP=$3
fi
DBUSER=test
DBPASS=xxxxxx

./tpcc_load -h $HOST -d $DBNAME -u $DBUSER -p $DBPASS -w $WH -l 1 -m 1 -n $WH >>
1.out &

x=1

while [ $x -le $WH ]
do
 echo $x $(( $x + $STEP - 1 ))
./tpcc_load -h $HOST -d $DBNAME -u $DBUSER -p $DBPASS -w $WH -l 2 -m $x -n $(( $x
+ $STEP - 1 ))  >> 2_$x.out &
./tpcc_load -h $HOST -d $DBNAME -u $DBUSER -p $DBPASS -w $WH -l 3 -m $x -n $(( $x
+ $STEP - 1 ))  >> 3_$x.out &
./tpcc_load -h $HOST -d $DBNAME -u $DBUSER -p $DBPASS -w $WH -l 4 -m $x -n $(( $x
+ $STEP - 1 ))  >> 4_$x.out &
 x=$(( $x + $STEP ))
done
```

tpcc初始化脚本

```
mysql -utest -pxxxxxx tpcc -hhost8_1 <create_table.sql
bash ./load_warehouse_para.sh tpcc 100 10

echo "mysql -utest -pxxxxxx tpcc -hhost8_1 <add_fkey_idx.sql"
```

tpcc测试语句

```
./tpcc_start -h host8_1 -P3306 -d tpcc -u test -pxxxxxx -w100 -c 100 -r 300 -l
3600 -i 10 -f ~/tpcc_proxy_mysql_t100_3600s.log
./tpcc_start -h host6 -P3306 -d tpcc -u test -pxxxxxx -w100 -c 100 -r 300 -l 3600
-i 10 -f ~/tpcc_proxy_t100_3600s.log
./tpcc_start -h host5 -P3440 -d tpcc -u test -pxxxxxx -w100 -c 100 -r 300 -l 3600
-i 10 -f ~/tpcc_proxy_lvs_t100_3600s.log
```