ProxySQL 双写功能测试与分析

Posted on 2017-08-09 by 周伟

1.扯淡.
谈到proxysql，想必在mysql圈混，应该不会陌生。那么双写什么？说白了就是应用每一次发到proxysql的请求，都会在两个数据库执行，这两个数据库是你指定的。proxysql官方叫Mirroring,换做中文，我们还是叫双写好一点。

大家首先想到的应该是替代mysql原生复制，应用直接写两份，两个库不用搭建主从复制了。这么想也没错，事情就是这样，但是用途有点不对，为啥呢？因为proxysql的双写并不能保证两个都写成功，也就是说这是个异步双写。一般我们用到哪些场景？首先，异步双写用于DB环境迁移，升级，验证SQL正确性。在不影响正式流量的前提下，搞一部分影子流量出来，写到另外一个库，完全不影响正式流量。其次，我们可以用双写功能让冷启动的库的缓存热起来，比如新加一个从库，如果直接切流量到新从库，肯定在刚开始一段时间DB的RT时间非常高，数据库性能抖动，对于许多应用这是没法接受的，如果切一部分影子流量，让Buffer pool暖起来，正式流量就可以平滑切到新库。

废话不多说，这里我们翻译了一篇percona的文章，有兴趣看看。

原文请看这里。

What About ProxySQL and Mirroring?

2017年5月25日

在这篇博客中，我们将看看ProxySQL和双写（mirroring）如何放在一起玩转。

概览

我个人非常喜爱这个ProxySQL，它是灵活扩展架构和高可用的重要组件。但不是所有会发光的都是黄金！在这篇文章中，我将尽力避免把“黄金”当作“碳”的事情发生（当然，碳有碳的用处，黄金有黄金的用处）。

FOR DBA 介绍

For DBA 是一个热爱数据库，Linux的团队，曾就职于某省级运营商，现就职于某某大型互联网公司。热衷于研究和分享各种Oracle，MySQL，MongoDB，以及Linux的使用经验和心得。

QQ交流群:545621153

标签

近期评论

文章归档

首先，我们需要介绍ProxySQL如何管理*流量调度*的基础知识 （我不想将其称为mirroring，下面将进一步解释）。

ProxySQL从应用程序接收一个连接，通过它可以进行一个简单的SELECT或更复杂的事务。ProxySQL获取每个查询，将其传递给查询处理器，处理它们，识别查询是否已被镜像，复制ProxySQL内部对象的整个MySQL会话，并将其与镜像队列（指的是镜像线程池）相关联。如果池是空闲的（在并发线程集中有可用的线程插槽），则立即处理查询。如果没有，查询将停留在队列中。如果队列已满，查询将丢失。

无论从查询返回什么，都会放到到/dev/null，因此没有结果集传回给客户端。

服务器的整个处理过程不是没有性能消耗的。如果您检查CPU利用率，您将看到ProxySQL中的 “mirroring” 实际上使CPU利用率翻倍了。这意味着由于资源争用，服务器A上的流量（traffic）会受到影响。

总结一下，ProxySQL会会做以下事情：

1. 以不同的顺序发送要执行的查询
2. 完全忽略事务隔离
3. 在B上执行的查询数量不同于A
4. 增加服务器资源上的负载

这些特性，再加上在本文末尾提到的期望，很清楚的是，目前我们不能认为ProxySQL是将服务器A到服务器B的负载复制的有效机制。

就个人而言，我不认为ProxySQL开发团队（Rene：D）应该浪费时间来解决这个问题，因为在ProxySQL中还有许多其他的东西要处理和改进。

在与ProxySQL打过多次交到之后，并在镜像（mirroring，习惯上我们还是译作镜像，下同）方面做了深入的研究，我认为要么将其视为基本的流量调度程序。否则，就需要重新概念化。但是一旦我们清楚的认识到，ProxySQL“ *流量调度* ”（仍然不想称之为mirroring）仍然是一个非常有趣的功能，能使程序更加有价值——特别是它很容易设置。

以下测试结果应该有助于得到正确的答案。

测试很简单：在 Percona XtraDB Cluster 加载数据（load data），并使用ProxySQL在MySQL主从环境中复制这个load data操作。

- MySQL / Percona XtraDB Cluster：VM CentOS 7，4 CPU，3 GB RAM，附加存储
- ProxySQL：VM CentOS 7，8 CPU，8GB RAM

为什么我选择给ProxySQL更多的资源？因为我知道我需要一些需要更多内存和CPU去设置其他的选项。我想确保我没有从ProxySQL中获得与CPU和内存有关的任何问题。

用来添加负载的应用程序是一个Java应用程序。该应用程序位于 https://github.com/Tusamarco/blogs/blob/master/stresstool_base_app.tar.gz

我用于进行测试的整套都在这里：https://github.com/Tusamarco/blogs/tree/master/proxymirror.

我用了四个不同的表：

+----------------------+

| Tables_in_mirror |

+----------------------+

| mirtabAUTOINC |

| mirtabMID |

| mirtabMIDPart |

| mirtabMIDUUID |

好吧，让我们开始吧。请注意，下面的测试是有意义的测试。请参考整套测试集。

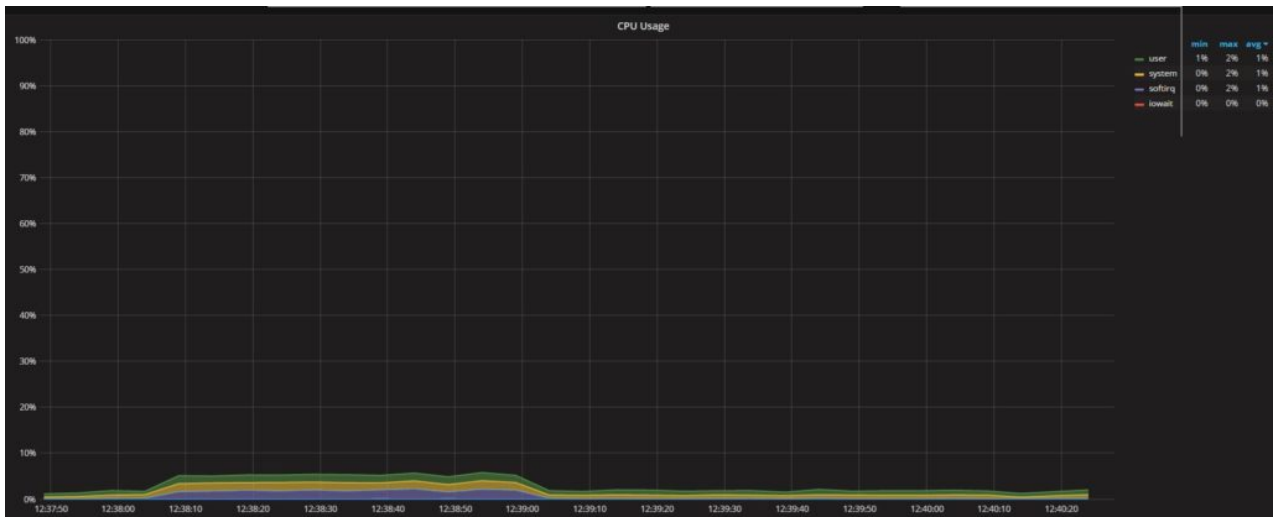
首先设置ProxySQL：

```
delete from mysql_servers where hostgroup_id in
(500,501,700,701);
INSERT INTO mysql_servers
(hostname,hostgroup_id,port,weight,max_connections)
VALUES ('192.168.0.5',500,3306,60000,400);
INSERT INTO mysql_servers
(hostname,hostgroup_id,port,weight,max_connections)
VALUES ('192.168.0.5',501,3306,100,400);
INSERT INTO mysql_servers
(hostname,hostgroup_id,port,weight,max_connections)
VALUES ('192.168.0.21',501,3306,20000,400);
INSERT INTO mysql_servers
(hostname,hostgroup_id,port,weight,max_connections)
VALUES ('192.168.0.231',501,3306,20000,400);
INSERT INTO mysql_servers
(hostname,hostgroup_id,port,weight,max_connections)
VALUES ('192.168.0.7',700,3306,1,400);
INSERT INTO mysql_servers
(hostname,hostgroup_id,port,weight,max_connections)
VALUES ('192.168.0.7',701,3306,1,400);
INSERT INTO mysql_servers
(hostname,hostgroup_id,port,weight,max_connections)
VALUES ('192.168.0.25',701,3306,1,400);
INSERT INTO mysql_servers
(hostname,hostgroup_id,port,weight,max_connections)
VALUES ('192.168.0.43',701,3306,1,400);
LOAD MYSQL SERVERS TO RUNTIME; SAVE MYSQL SERVERS TO
DISK;
delete from mysql_users where username='load_RW';
insert into mysql_users
(username,password,active,default_hostgroup,default_schem
a,transaction_persistent) values
('load_RW','test',1,500,'test',1);
LOAD MYSQL USERS TO RUNTIME;SAVE MYSQL USERS TO DISK;
delete from mysql_query_rules where rule_id=202;
insert into mysql_query_rules
(rule_id,username,destination_hostgroup,mirror_hostgroup,
active,retries,apply)
values(202,'load_RW',500,700,1,3,1);
LOAD MYSQL QUERY RULES TO RUNTIME;SAVE MYSQL QUERY RULES
TO DISK;
```

Test 1

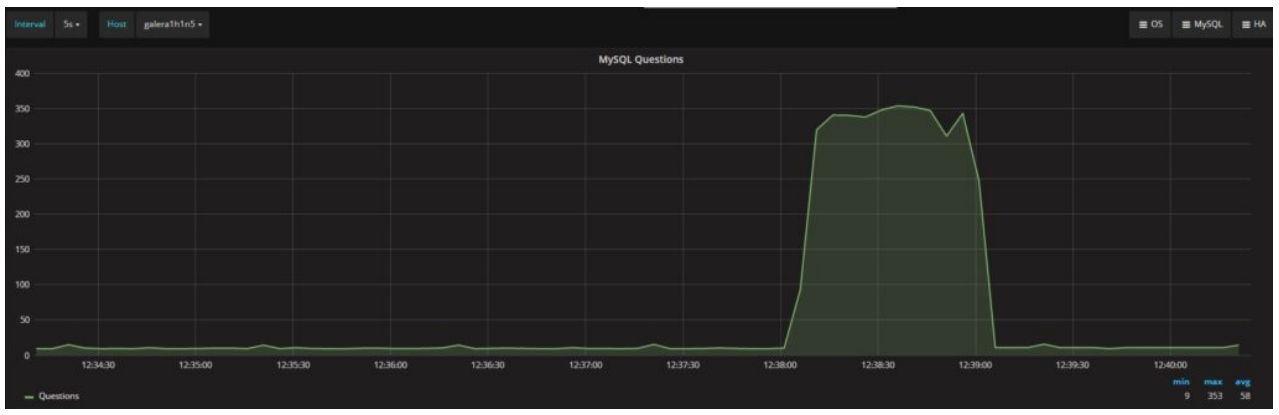
第一个测试主要是简单的功能测试，即在Percona XtraDB Cluster和MySQL中使用单个线程插入记录。有3000个循环，测试结束时，在两个平台上都有3000条记录。

画一条基线，我们可以看到ProxySQL CPU利用率相当低：

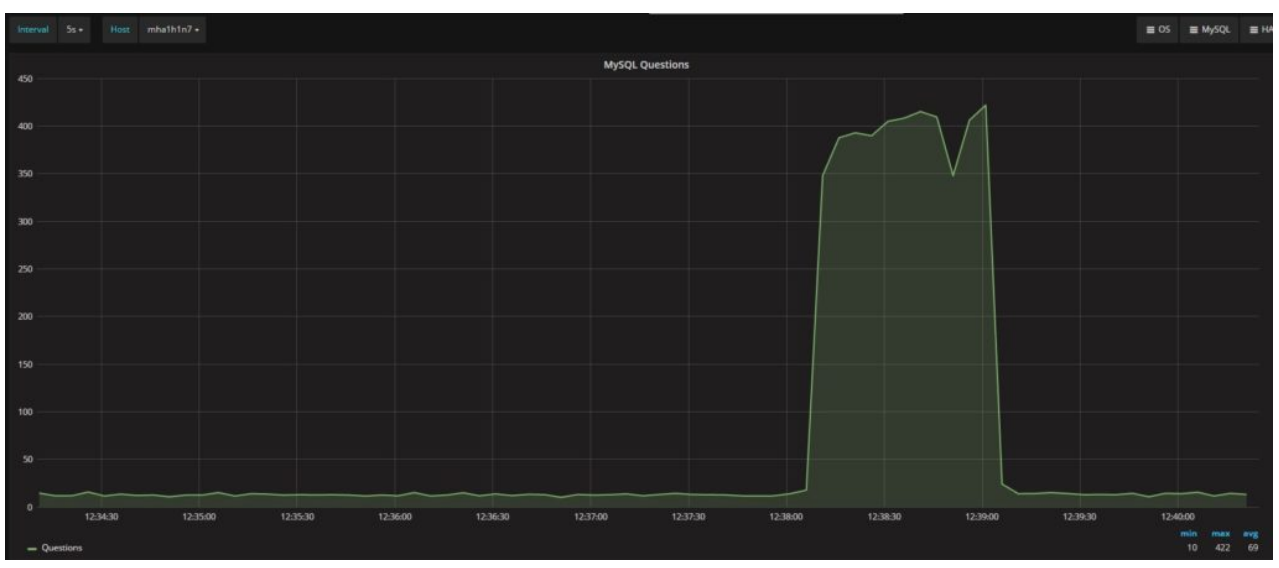


同时，Percona XtraDB Cluster和MySQL的“questions”数量也非常相似：

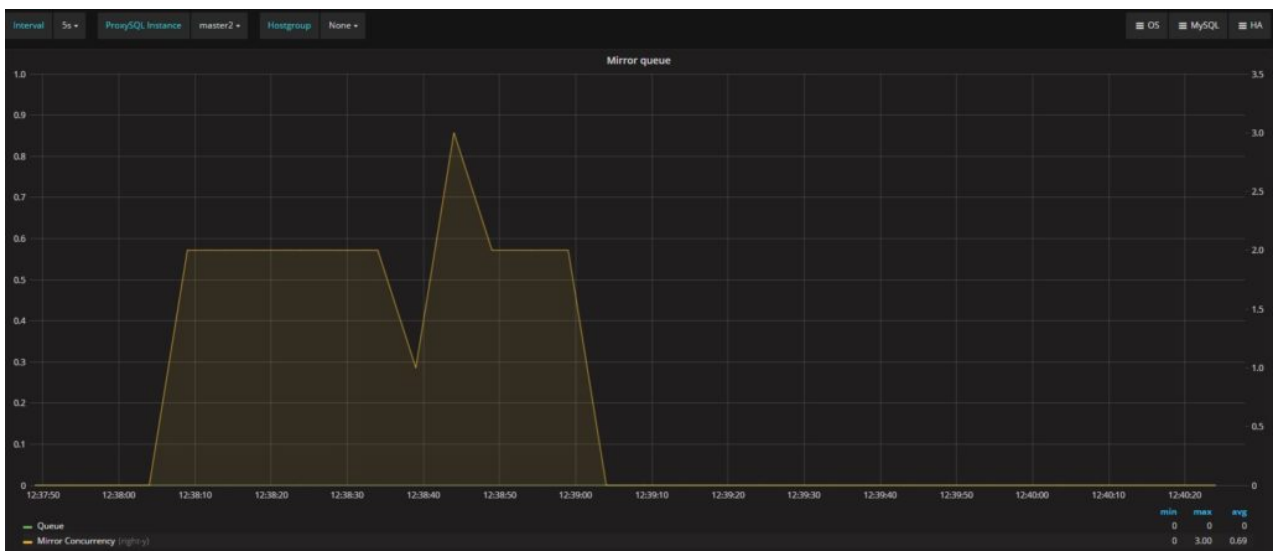
Percona XtraDB Cluster



MySQL



我们要关注的另外两个指标是Mirror_concurrency和Mirror_queue_length。这两个分别指向mysql-mirror_max_concurrency和mysql-mirror_max_queue_length：



ProxySQL 1.4.0中引入了这两个新变量和指标，目的是控制和管理ProxySQL内部产生的负载，且与*mirroring*功能相关。在这种情况下，可以看到我们最多有三个并发连接和零个队列条目（都不错）。

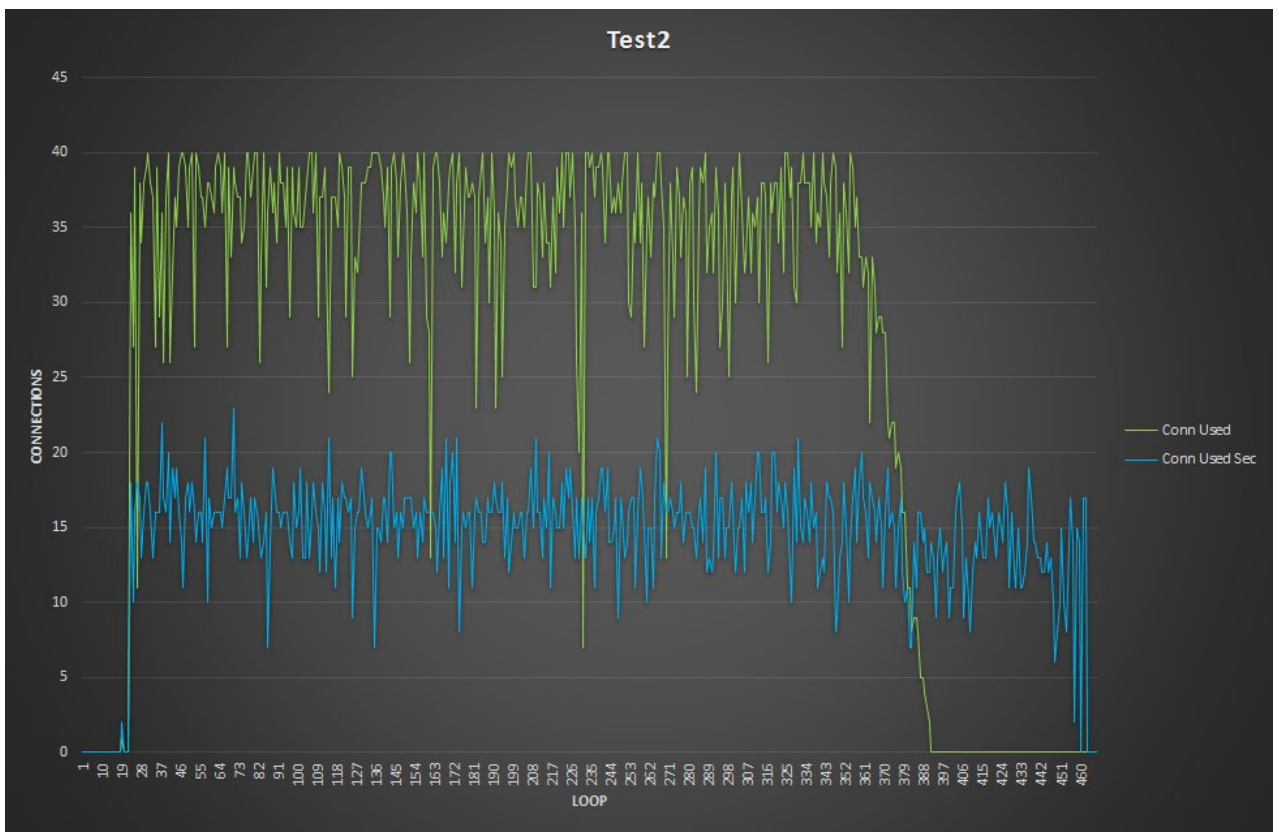
既然我们有了一个基准，而且我们在功能层面知道“它有效”，看看增加负载会发生什么。

Test 2

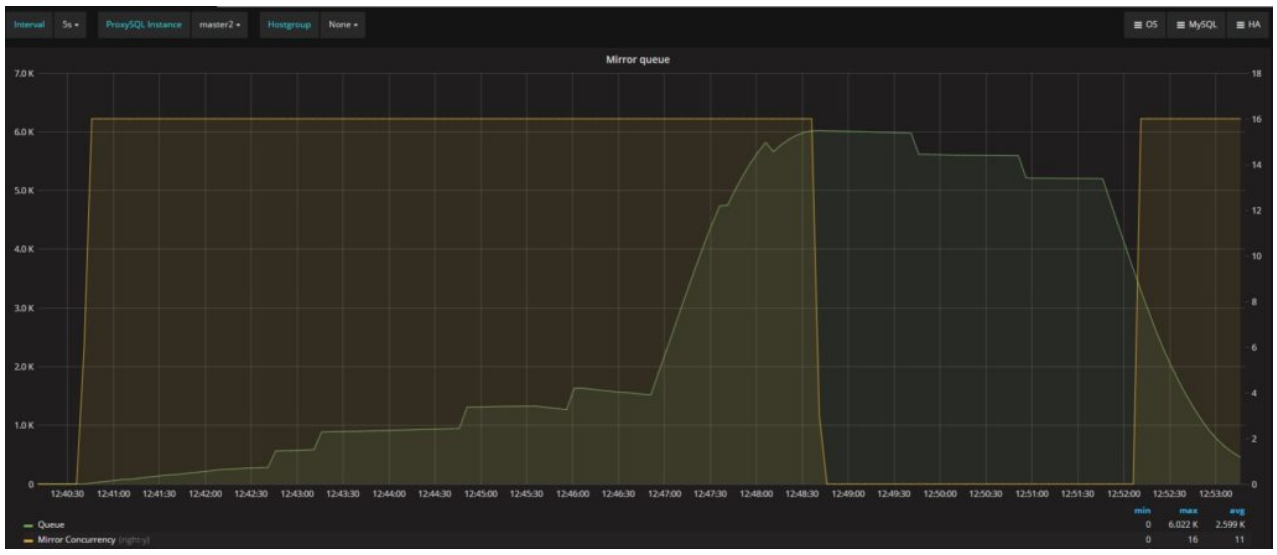
测试的范围是，当用标准的配置和不断增加的负载时，ProxySQL会表现出什么。结果表明，一旦ProxySQL有一点点负载，就开始丢失（lost）一些查询。

执行40个线程的3000个循环仅导致在Percona XtraDB Cluster中的所有四个表中插入了120,000行。但是，辅助（mirrored）平台中的表只有一个变量或插入的行，介于101,359和104,072之间。这表明数据的丢失是一致的。

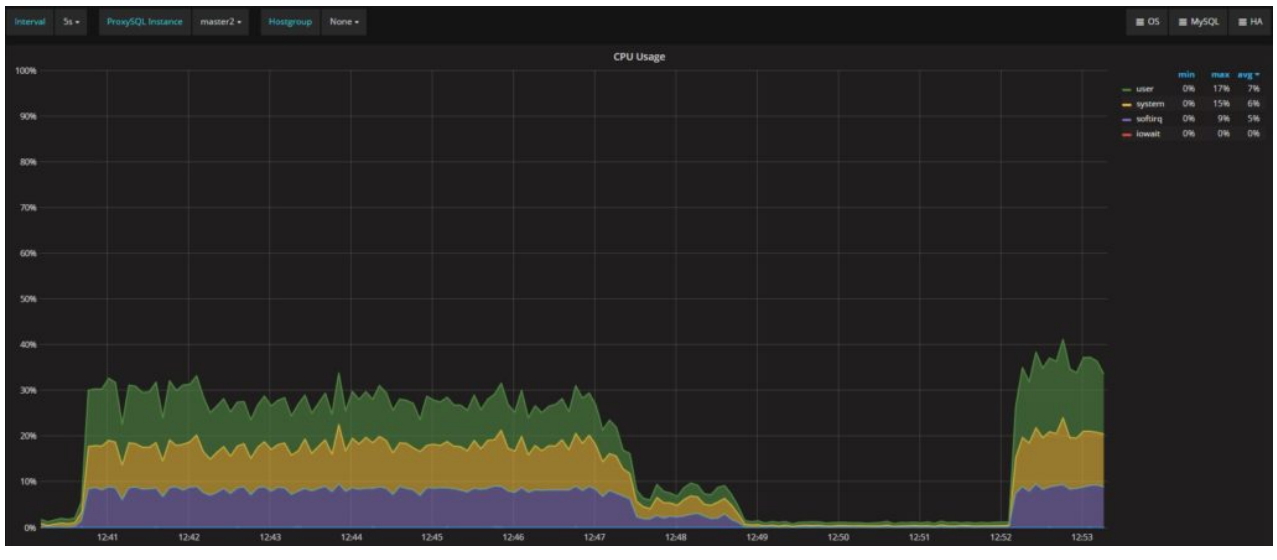
在查看并比较在Percona XtraDB Cluster和辅助服务器上运行的连接之后，我们可以看到（正如预期的那样）Percona XtraDB Cluster的连接数量正在变化并随着请求数量而变化，而辅助节点上的连接数为默认值：**mysql-mirror_max_concurrency = 16**。



有趣的是，ProxySQL事务进程队列保持与Secondary连接比与Percona XtraDB Cluster连接的时间更长。



由上图可知，队列是一个明显的铃声曲线（bell curve），达到6K个entries（这远低于**mysql-mirror_max_queue_length的**限制（32K））。然而，ProxySQL删除了查询，这表示队列不足以容纳待处理的查询。

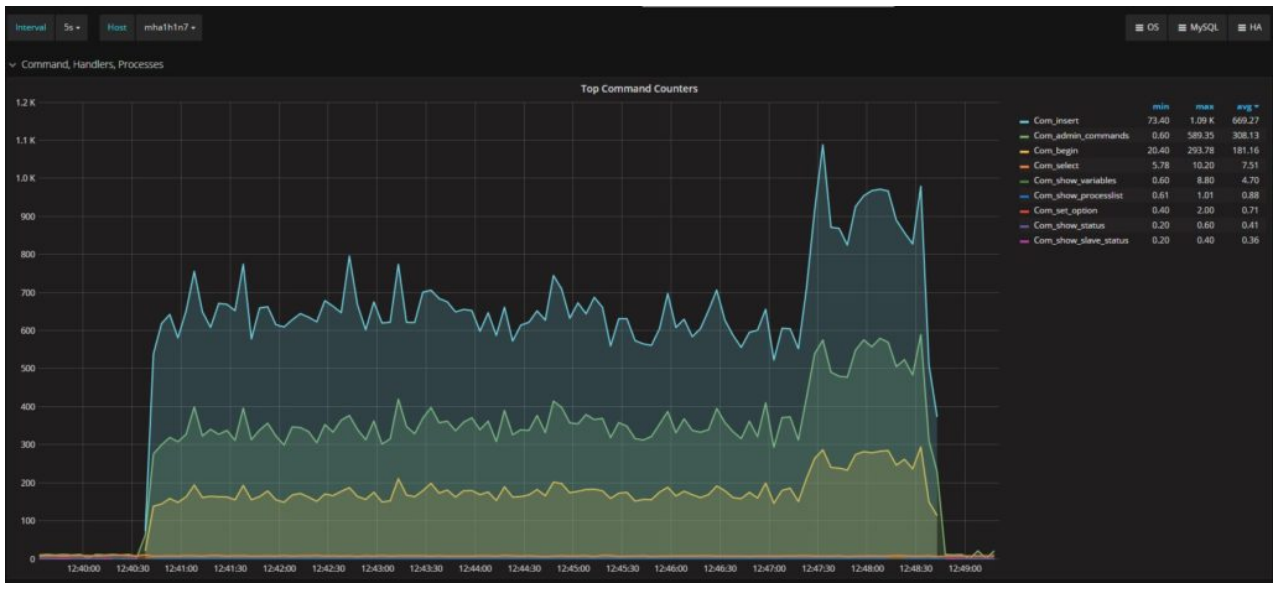


CPU-wise，ProxySQL（如预期的那样）需要一些的CPU周期，但没有什么疯狂。上图所示，在12:47左右，主要负载停止后，可以看出简单镜像队列处理的负载。

另一个需要注意的的是描述Percona XtraDB Cluster和secondary的执行命令的图：

Percona XtraDB Cluster

Secondary



正如你所看到的，secondary 上的流量明显更少（平均669，与Percona XtraDB Cluster的1.17K相比）。然后，当Percona XtraDB Cluster节点的主要负载终止时，它会达到顶峰。简而言之，很明显的是，ProxySQL无法按照Percona XtraDB Cluster中存在的流量进行扩展，实际上在secondary上丢失了大量的数据。(In short it is quite clear that ProxySQL is not able to scale following the traffic existing in Percona XtraDB Cluster, and actually loses a significant amount of data on the secondary.)

在测试3中，使负载加倍也显示了相同的结果，ProxySQL达到了流量重复的限制。（with ProxySQL reaches its limit for traffic duplication.）

那么可以优化吗?

答案当然是肯定的！这就是 mysql-mirror_max_concurrency 存在的价值，如果将它的值从16增加到100（只是为了使它变得很高）看看会发生什么。

Test 4 (two app node writing)

首先，Percona XtraDB Cluster和secondary 表中的需要有相同的行数（24万）。这是一个前提。（That is a good first win.）

第二，注意运行的连接数：

The graphs are now are much closer, and the queue drops to just a few entries.

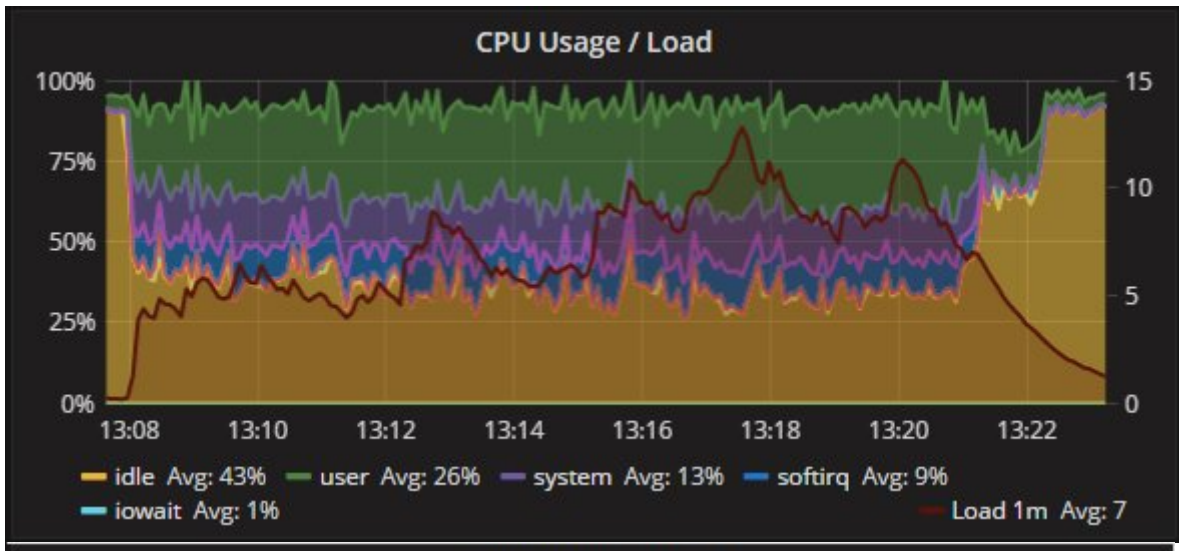
Percona XtraDB Cluster 中执行的命令：

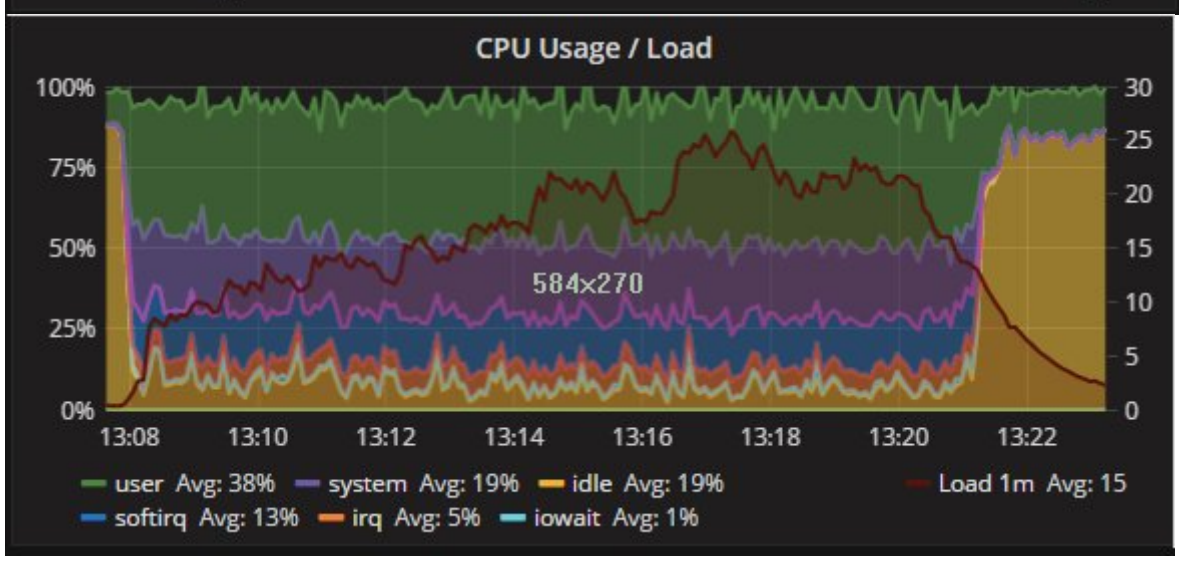
secondary 中执行的命令：

平均执行报告了相同的值，趋势也非常相似。（Average execution reports the same value, and very similar trends.）

最后看看，CPU的消耗和影响是什么？

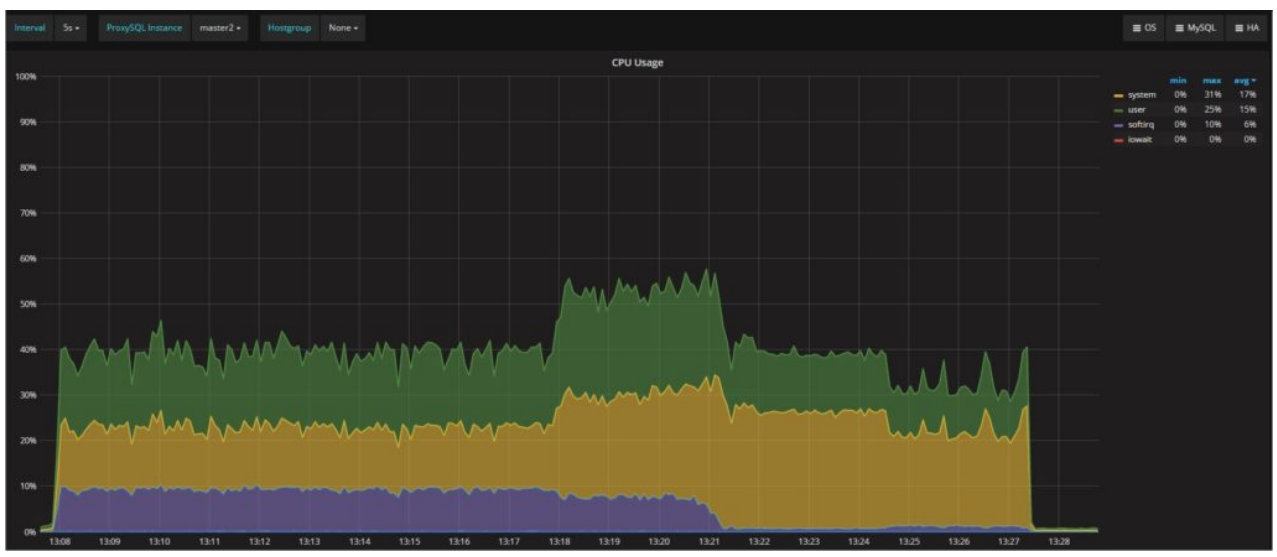
Percona XtraDB Cluster和secondary 的CPU利用率：





如预期的那样，存在CPU使用分配的一些差异。但是两个节点之间的趋势是一致的，节点之间的操作也是一致的。

ProxySQL CPU利用率明显高于以前：

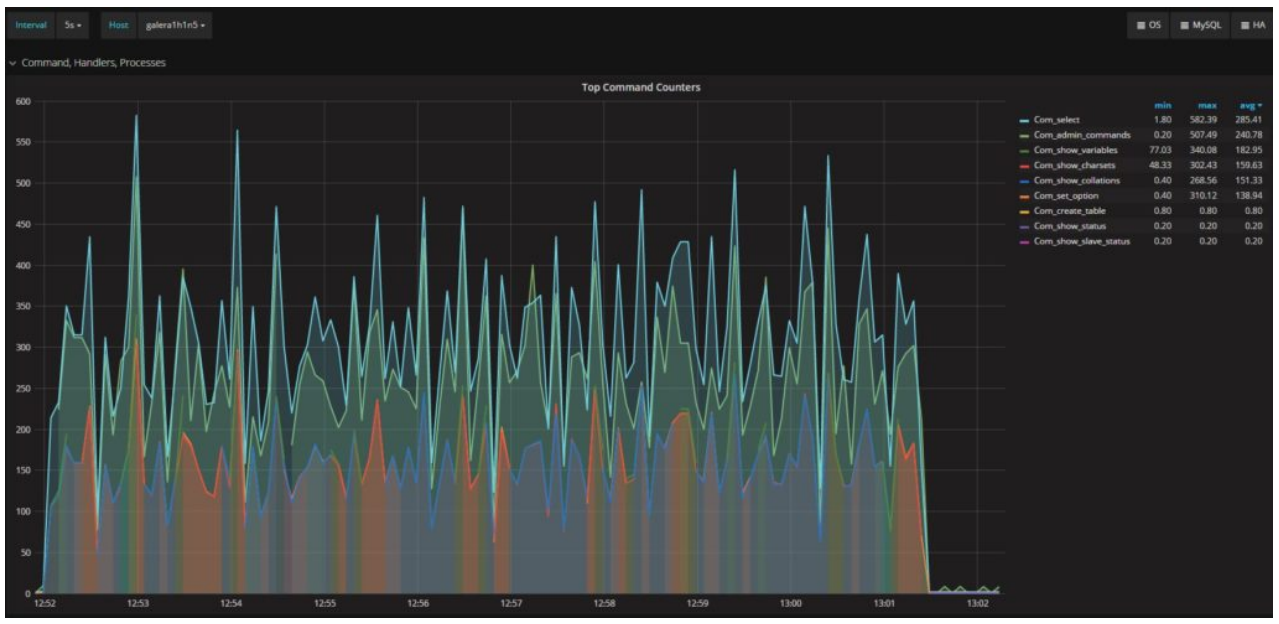


但它是绝对可控的，且仍然反映了初始分配。

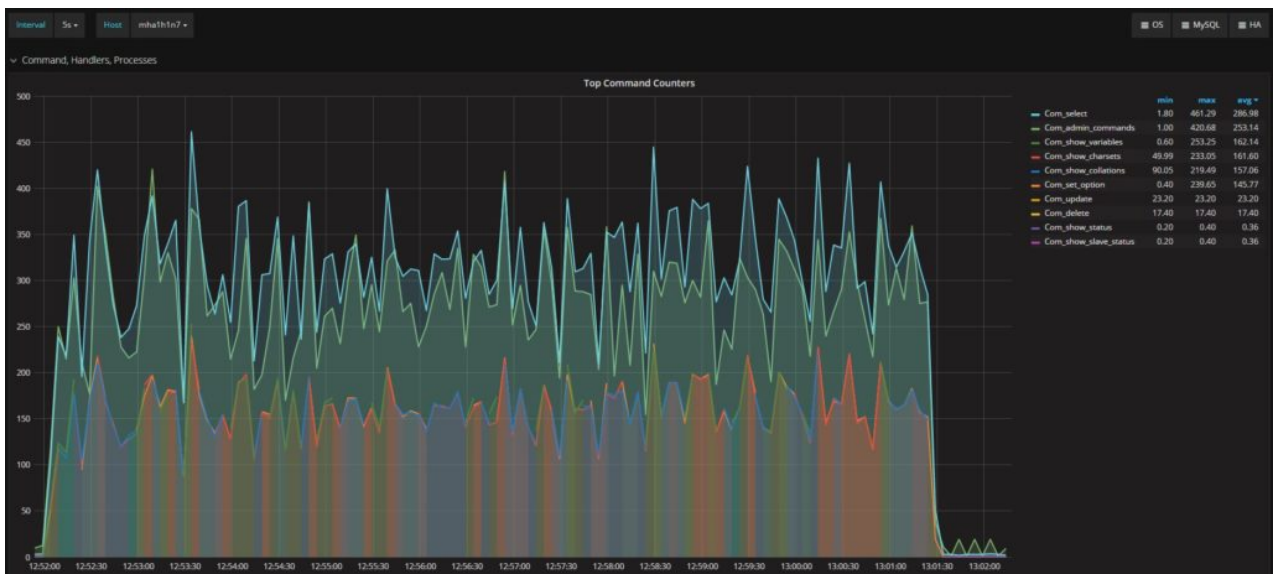
那么，CRUD 操作又会怎么样呢？到目前为止，我只测试了insert操作，但是如果我们运行一个完整的CRUD测试集，会发生什么？

Test 7 (CRUD)

首先，我们回过头来看看Percona XtraDB Cluster中执行的命令：



secondary：

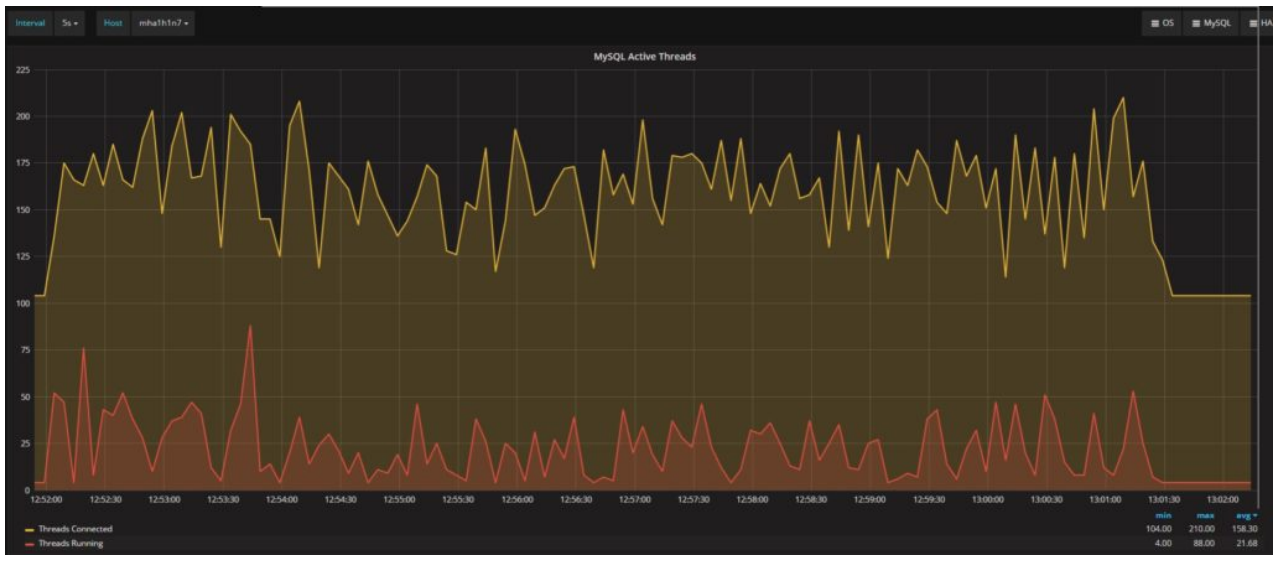


在外观上，我们有非常相似的工作负载，选择行为会显着分歧。这是因为在secondary 中不同的操作不被事务封装。他们按照接收的那样被执行。我们可以看到两者之间的更新和删除操作有显着差异。

此外，执行中的线程在两个平台之间显示不同的结果：

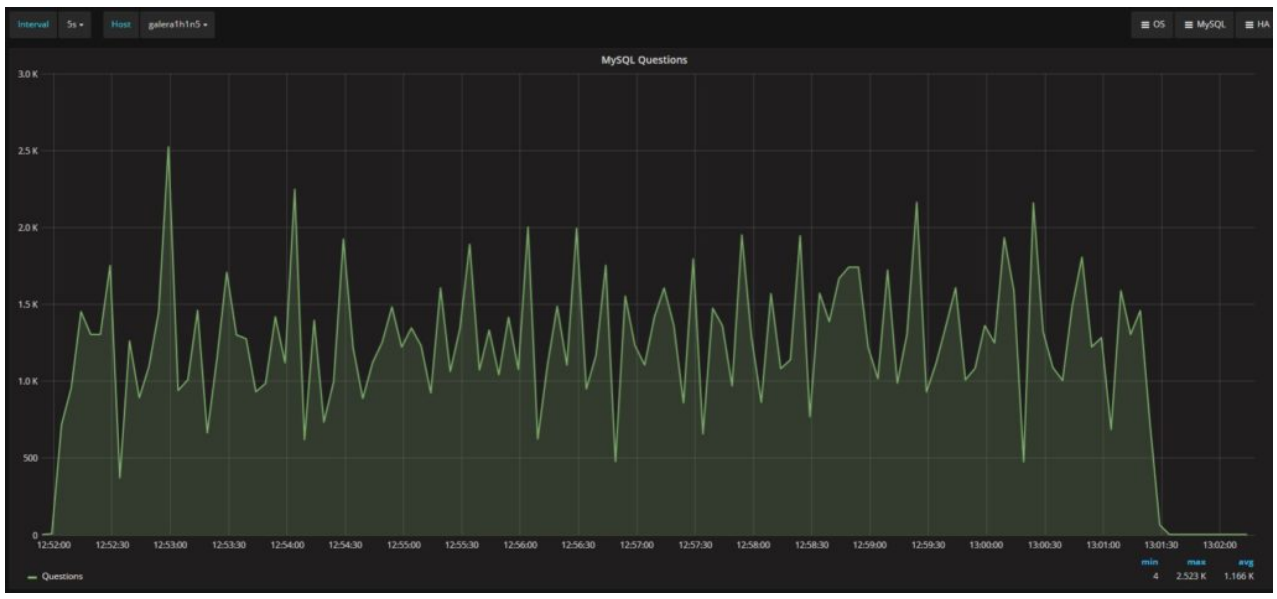
Percona XtraDB Cluster

Secondary

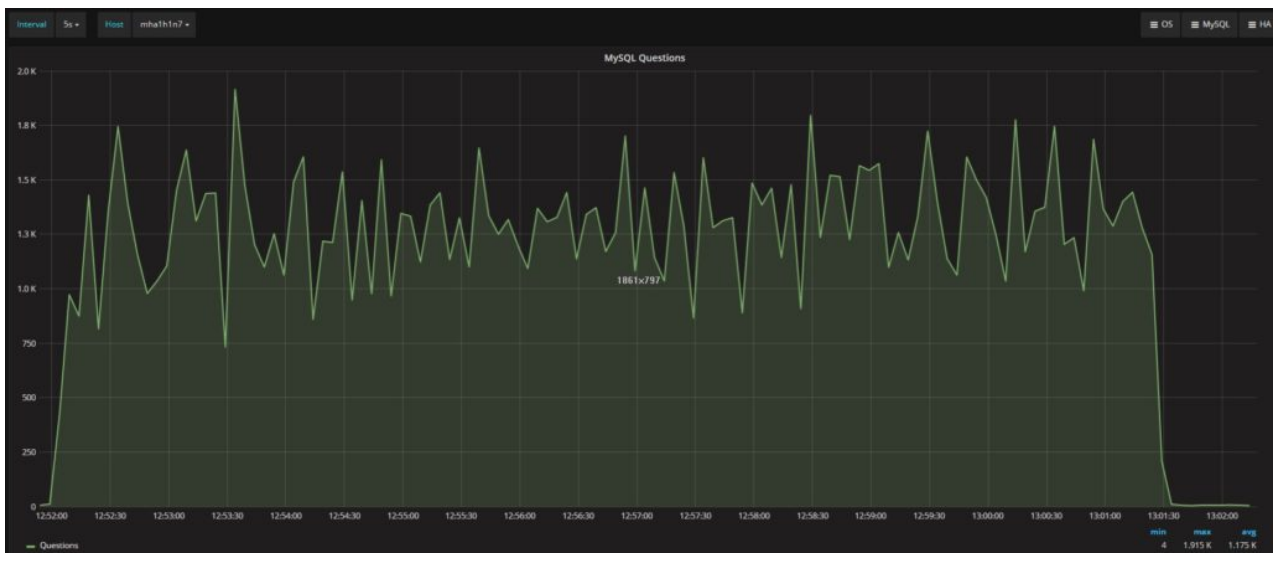


很明显，Percona XtraDB Cluster不断运行更多的线程和更多的连接。然而，两个平台处理的问题总数相似：

Percona XtraDB Cluster



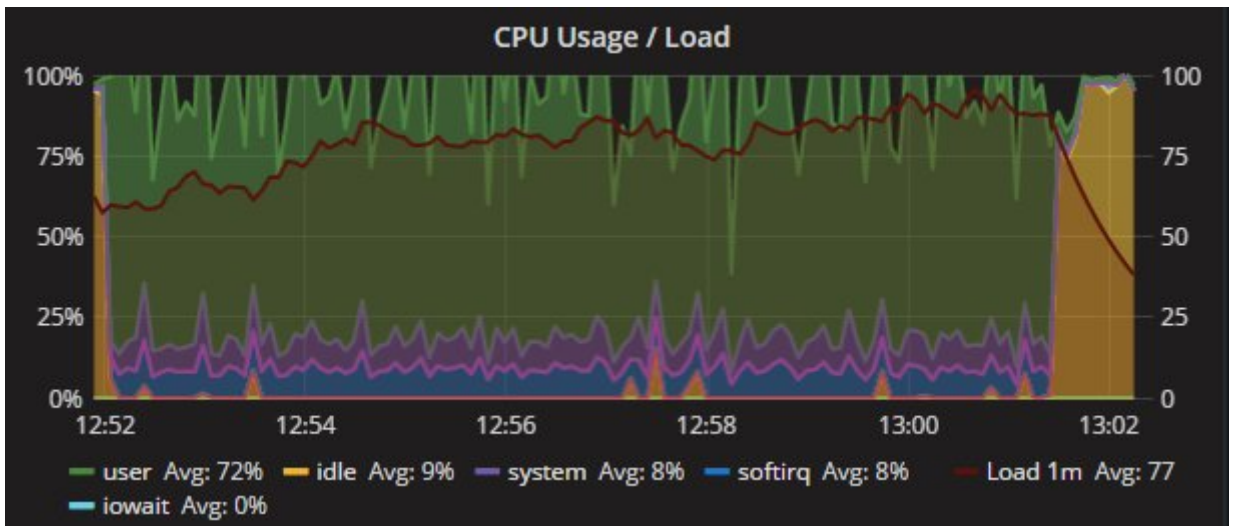
Secondary



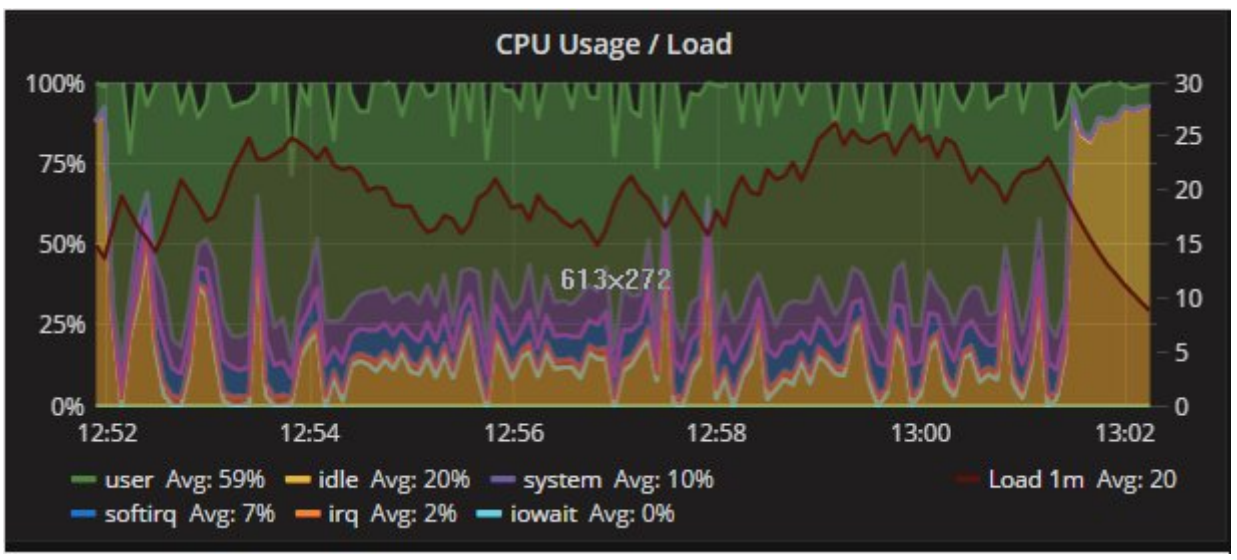
两者的平均值都在1.17K /s (questions) 左右。

这也是另外一个迹象表明并发操作的影响有多少，而不考虑隔离或执行顺序。）（This is also another indication of how much the impact of concurrent operation on behavior, with no respect to the isolation or execution order.）下面我们可以通过查看CPU利用率来清楚地看到不同的结果：

Percona XtraDB Cluster



Secondary



结论

在结束这篇文章之前，让我们回到文章的开始。我们不能将ProxySQL中的*mirror*功能视为真正的mirroring，更多的是流量重定向（请去这里here查看我关于mirroring 更多的分析）。

以这种方法用ProxySQL在测试负载及其在secondary 平台上的效果方面仍然部分有效。我们知道，在这种情况下不能保证数据一致性，并且select，update和delete会受到影响（给定他们不同的数据集和结果集）。

服务器的行为会在原始和镜像之间发生变化，如果不是在数量或质量方面。（The server behaviors change between the original and mirror, if not in the quantity or the quality.）

我相信，当我们需要一个能够在不同或新的平台上测试我们的生产负载的工具时，我们会更好地寻找别的东西。可能会是 *query Playback*，最近由DropBox 进行了检查并修补。（https://github.com/Percona-Lab/query-playback）

最后想说的是，ProxySQL已经是一个很酷的工具。如果它不能很好的cover mirroring ，我还可以忍受。我有志于使它起到它应该有的作用（它在许多其他功能上也一样）。

致谢

像往常一样，需要感谢 Rene ——致力于修复和引入与*mirroring*相关联的新功能 ，如队列和并发控制。

另外，对 Percona 开发 Percona Monitoring and Management (PMM) 的团队表示感谢，所有的图表（3除外）都来自PMM（其中一些是我定制的）。

注：本文译自Percona官方的一篇博客，其中标出原文的地方译者自觉翻译不佳，欢迎访客留言指正。

Posted in MySQL

Previous:
10分钟让你明白MySQL是如何利用索引的

Next:
pt-online-schema-change MySQL在线DDL利器