13 学会对MySQL做基准测试，掌握数据库性能

更新时间：2020-03-30 10:13:22

> 学习要注意到细处，不是粗枝大叶的，这样可以逐步学习、摸索，找到客观规律。 —— 徐特立

不论你是做哪个方向，使用哪种框架或工具，对于你的最终作品，都一定离不开基准测试。简单的说，基准测试就是要看看产品的性能是否符合预期，或达到某种预定义的指标。MySQL 作为明星级存储工具，应用范围之广不必多说。为了预见 MySQL 的可用性，我们就需要对 MySQL 做基准测试。这一节里，我会先去讲解基准测试的理论，再去介绍 sysbench（基准测试工具），最后，演示说明针对 MySQL 的基准测试。

1 关于基准测试，你需要知道这些

通常，我们去认识一种新的技术，都会带着三个问题去学习并思考：它是什么 ？它能够做什么 ？又应该怎样使用它呢 ？接下来，我会从理论层面去解答前两个问题。

1.1 什么是基准测试

基准测试可以理解为对系统的一种压力测试（也就是常说的压测），但是它又与压测有着显著的区别。压测通常都是需要考虑业务逻辑的，毕竟它是业务上线之前的最后一道工序，需要使用真实的数据（或近似数据）。而基准测试则宽泛很多，它不要求去关注业务实现，数据可以由工具生成，所以，更加简单、直接。

而对于数据库的基准测试，它其实是有定义的：

> 数据库的基准测试是对数据库的性能指标进行定量的、可复现的、可对比的测试。

1.2 基准测试的作用

如果你经常接触业务系统，你就一定知道，系统的性能瓶颈基本都在 IO 上。IO 又会分为两类：磁盘 IO 和网络 IO。对于网络 IO 来说，不确定性太大了，单独的一方很难去做优化。所以，大多数的性能优化都是针对磁盘 IO 的，而这里的磁盘 IO 指的就是数据库的读写操作。

对数据库的基准测试，就是要分析在当前的环境配置（硬盘配置、操作系统配置、数据库配置等等）下数据库的性能表现。并通过不断的调校配置和测试过程，来找到 MySQL 的最佳性能阈值，提供高可用的环境。

1.3 基准测试的指标

从出现基准测试，再到逐步发展，关于测试的指标是“与日俱增”的。但实际上，常用的数据库指标就那么几个（这样的情况可以引申的案例简直不胜枚举），一起来看一看吧。

- TPS：Transactions Per Second 的首字母缩写，每秒处理完成的事务次数
- QPS：Query Per Second 的首字母缩写，每秒查询次数，与 TPS 一样，都是用来衡量系统吞吐量
- 响应时间：包含平均、最大、最小响应时间
- 并发量：可以同时处理的请求数量

在对 MySQL 做基准测试时，一定要去选择专业的工具，不要重复造轮子。sysbench 通用且功能强大，最让人欣喜的是：它非常适合 InnoDB，因为它模拟了许多 InnoDB 的特性。接下来，我们就去学习下 sysbench。

2 基准测试工具 sysbench

sysbench 确实是一款非常不错的数据库性能测试工具，起码用过的人都说好。下面，我不仅要告诉你 sysbench 能做什么，还要带着你把它安装到你的机器中。最后，讲解它的语法和使用说明。

2.1 初识 sysbench

sysbench 是跨平台的基准测试工具，支持多线程和多种数据库（Mysql、PostgreSQL、Oracle 等等）。它的测试能力非常广，主要包括：

- CPU 运算性能
- 磁盘 IO 性能
- 内存分配及传输速度
- POSIX 线程性能
- 数据库性能（OLTP 基准测试）

这里需要注意，在谈到测试数据库性能的时候，说的是针对 OLTP（另一类是 OLAP）的基准测试。这里大家不需要担心，因为大多数数据服务都是 OLTP 类型的（可以动手查一查什么是 OLTP，什么又是 OLAP）。

2.2 安装 sysbench

关于 sysbench 的安装方法，在官网中已经给出了详细的说明（如果你的英文比较好，也可以去读一读官网的说明）。sysbench 能够很好的支持 Linux 和 macOS，对于 Windows 来说可能就差强人意了。下面，我将给出在 Linux 和 macOS 上的安装方法说明：

```
# Debian/Ubuntu
curl -s https://packagecloud.io/install/repositories/akopytov/sysbench/script.deb.sh | sudo bash
sudo apt -y install sysbench

# RHEL/CentOS
curl -s https://packagecloud.io/install/repositories/akopytov/sysbench/script.rpm.sh | sudo bash
sudo yum -y install sysbench

# Fedora
curl -s https://packagecloud.io/install/repositories/akopytov/sysbench/script.rpm.sh | sudo bash
sudo dnf -y install sysbench

# macOS
brew install sysbench
```

如果你的机器是 mac，那就会非常的方便了（不仅仅是 sysbench，对于其他工具的安装也非常方便），只需要通过 brew 就可以完成安装。安装完成之后，可以验证下是否安装成功：

```
# 直接在终端下执行即可（由于存在版本更新的可能，你的版本号可能与我的不同）
□ ~ sysbench --version
sysbench 1.0.18
```

2.3 sysbench 语法

只需要在终端下执行 `sysbench --help` 命令，就可以看到关于 sysbench 的详细使用方法。首先，看一看它的基本语法：

```
sysbench [options]... [testname] [command]
```

所以，你也就知道了，学会使用 sysbench，只需要搞明白：options、testname、lua_options 和 command。options 是选项或参数的意思，它涵盖的内容非常多，例如：数据库相关的、日志相关的、行为相关的等等。下面，我先去介绍一些常用的 options：

- 通用数据库选项
  - –db-driver：指定数据库的类型，默认是 mysql
- MySQL 相关选项
  - –mysql-host：服务器主机地址，默认是 localhost
  - –mysql-port：服务器端口，默认是 3306
  - –mysql-user：用户名，默认是 sbtest
  - –mysql-password：密码
  - –mysql-db：指定测试数据库，默认是 sbtest（需要自行创建）
- 执行（通用）选项
  - –threads：创建测试线程的数量，默认是1
  - –events：事件数量

- –time：最大执行时间，单位是秒
- –report-interval：生成报告的时间间隔，0代表禁止，默认是0
- –debug：是否打印更多的调试信息，默认是 off
- –config-file：配置文件

testname 标识要进行的测试，在老版本的 sysbench 中，可以通过 --test 选项指定测试脚本，而在新版本中，–test 已经被废弃了，而是直接可以指定测试脚本。例如：

```
 ~ sysbench /usr/local/Cellar/sysbench/1.0.18_1/share/sysbench/oltp_read_only.lua
```

sysbench 测试使用的是 lua 脚本，绝大多数情况下，使用 sysbench 自带的就已经足够了。对于不同版本的 sysbench，脚本的位置也可能会不同。下面，我给出一种简单的方法来寻找这些自带的脚本：

```
# 先确定你已经成功安装了 sysbench
 ~ sysbench --version
sysbench 1.0.18

# 使用 which 命令找到可执行文件的位置
 ~ which sysbench
/usr/local/bin/sysbench

# 可以发现 /usr/local/bin/sysbench 是个软链接，同时也指出了 sysbench 的安装目录
 ~ ls -l /usr/local/bin/sysbench
lrwxr-xr-x 1 qinyi 40 12  3 20:19 /usr/local/bin/sysbench -> ../Cellar/sysbench/1.0.18_1/bin/sysbench

# 进入到 sysbench 的安装目录中
 ~ cd /usr/local/Cellar/sysbench/1.0.18_1/

# 自带的 lua 脚本位于安装目录下的 share/sysbench 目录中
 1.0.18_1 ls -lt share/sysbench
total 60
drwxr-xr-x 5 qinyi   160 12  3 20:19 tests
-rwxr-xr-x 1 qinyi  1452 10 21 14:23 bulk_insert.lua
-rw-r--r-- 1 qinyi 14369 10 21 14:23 oltp_common.lua
-rwxr-xr-x 1 qinyi  1290 10 21 14:23 oltp_delete.lua
-rwxr-xr-x 1 qinyi  2415 10 21 14:23 oltp_insert.lua
-rwxr-xr-x 1 qinyi  1265 10 21 14:23 oltp_point_select.lua
-rwxr-xr-x 1 qinyi  1649 10 21 14:23 oltp_read_only.lua
-rwxr-xr-x 1 qinyi  1824 10 21 14:23 oltp_read_write.lua
-rwxr-xr-x 1 qinyi  1118 10 21 14:23 oltp_update_index.lua
-rwxr-xr-x 1 qinyi  1127 10 21 14:23 oltp_update_non_index.lua
-rwxr-xr-x 1 qinyi  1440 10 21 14:23 oltp_write_only.lua
-rwxr-xr-x 1 qinyi  1919 10 21 14:23 select_random_points.lua
-rwxr-xr-x 1 qinyi  2118 10 21 14:23 select_random_ranges.lua
```

command 是 sysbench 要执行的命令，一共有四类：

- prepare：准备数据的命令。在 sysbench 压力测试之前，需要先准备好测试库、测试表以及测试表中的数据
- run：进行压力测试
- cleanup：清除测试时产生的数据
- help：输出给定 lua 脚本的帮助信息

2.4 sysbench 使用说明

至此，已经介绍了 sysbench 的安装和使用方法，那么，接下来就一定是使用 sysbench 做基准测试了。别着急，先来看看关于 sysbench 的使用说明：

- 不要在线上 MySQL 服务器执行测试，这会严重影响服务性能
- 逐步增加并发连接数（–threads 选项，例如：10、20、50、100），观察 MySQL 的表现
- 每一轮完整的测试都应该包含 cleanup

3 使用 sysbench 对 MySQL 进行测试

sysbench 的测试过程其实就是对应于它的 command（prepare、run、cleanup），那么，我接下来就按照这个步骤完整的走一遍对 MySQL 的基准测试。最后，再去解读测试报告以及给出测试建议。

3.1 测试前的准备工作

由于 sysbench 使用 sbtest 库（默认使用的库名）做测试，我们就需要先来创建一个测试库（当然也可以自行指定其他的库，但必须是要存在的）：

```
# 通过 mysqladmin 工具创建 sbtest 测试库（通过 MySQL 客户端 CREATE DATABASE 也是一样的）
 ~ mysqladmin -h127.0.0.1 -uroot -proot -P3306 create sbtest
mysqladmin: [Warning] Using a password on the command line interface can be insecure.
```

创建测试库之后，就可以使用 sysbench 去完成准备工作了。我们在 sbtest 库中创建5张测试表，每张表中插入5000条数据，执行如下命令：

```
# --tables=5表示创建5个测试表，--table_size=5000表示每个表中插入5000行数据
 ~ sysbench --mysql-host=127.0.0.1 \
      --mysql-port=3306 \
      --mysql-user=root \
      --mysql-password=root \
      /usr/local/Cellar/sysbench/1.0.18_1/share/sysbench/oltp_common.lua \
      --tables=5 \
      --table_size=5000 \
      prepare
sysbench 1.0.18 (using bundled LuaJIT 2.1.0-beta2)

Creating table 'sbtest1'...
Inserting 5000 records into 'sbtest1'
Creating a secondary index on 'sbtest1'...
Creating table 'sbtest2'...
Inserting 5000 records into 'sbtest2'
Creating a secondary index on 'sbtest2'...
Creating table 'sbtest3'...
Inserting 5000 records into 'sbtest3'
Creating a secondary index on 'sbtest3'...
Creating table 'sbtest4'...
Inserting 5000 records into 'sbtest4'
Creating a secondary index on 'sbtest4'...
Creating table 'sbtest5'...
Inserting 5000 records into 'sbtest5'
Creating a secondary index on 'sbtest5'...
```

由于之前已经讲解过这些选项、参数的含义，我这里就不再赘述了（命令输出也比较简单、易读）。准备工作完成之后，你可以去数据库中查看下 sysbench 都 fake 了哪些数据。

3.2 执行测试

如果你仔细观察 sysbench 自带的 lua 脚本名称，你就会发现，它们包含了很多类型的测试，例如：只读测试、只写测试、删除测试、大批量插入测试等等。我们这里挑选 oltp_read_write.lua（任意挑选，不具有特殊含义）来执行测试：

```
# 测试 60s，并把测试输出到 /tmp/sysbench.log 文件中
~ sysbench --threads=4 \
    --time=60 \
    --report-interval=10 \
    --mysql-host=127.0.0.1 \
    --mysql-port=3306 \
    --mysql-user=root \
    --mysql-password=root \
    /usr/local/Cellar/sysbench/1.0.18_1/share/sysbench/oltp_read_write.lua \
    --tables=5 \
    --table_size=5000 \
    run >> /tmp/sysbench.log
```

执行上面这个命令，我们的终端会 hang 住60秒，并把打印输出重定向到 /tmp/sysbench.log 文件中。这里我暂时不去解读这份测试输出，不过，你可以先打开去看一看。

3.3 测试后的清理工作

在介绍 sysbench 的时候我就说过，每一轮完整的测试过程都应该包含清理工作。所以，当执行完上面的命令之后，紧接着执行下 cleanup 吧：

```
~ sysbench --mysql-host=127.0.0.1 \
    --mysql-port=3306 \
    --mysql-user=root \
    --mysql-password=root \
    /usr/local/Cellar/sysbench/1.0.18_1/share/sysbench/oltp_common.lua \
    --tables=5 \
    cleanup
sysbench 1.0.18 (using bundled LuaJIT 2.1.0-beta2)

Dropping table 'sbtest1'...
Dropping table 'sbtest2'...
Dropping table 'sbtest3'...
Dropping table 'sbtest4'...
Dropping table 'sbtest5'...
```

可以看到，cleanup 过程是非常简单的，它直接把创建的5张测试表 DROP 掉了。不过，需要注意，cleanup 并没有删除测试库 sbtest。我们可以据此得出结论，sysbench 不会参与到测试库的创建和删除过程。

3.4 测试结果的分析

在执行 run 命令时，我将结果输出重定向到了 /tmp/sysbench.log 文件中，这里，我们来一起解读下，看看基准测试的结果（已经做了相关注释）。

```
sysbench 1.0.18 (using bundled LuaJIT 2.1.0-beta2)

# 执行测试的一些基本设置：并发线程数为4、每10秒报告一次测试结果
Running the test with following options:
Number of threads: 4
Report intermediate results every 10 second(s)
Initializing random number generator from current time

# 初始化（测试）工作线程
Initializing worker threads...

Threads started!

# 每10s报告一次测试结果
# thds（线程数）、tps（每秒事务数）、qps（每秒查询数）、r/w/o（每秒的读/写/其它次数）、lat（延迟）、err/s（每秒错误数）、reconn/s（每秒重连次数）
[ 10s ] thds: 4 tps: 93.38 qps: 1875.13 (r/w/o: 1312.87/375.10/187.15) lat (ms,95%): 58.92 err/s: 0.00 reconn/s: 0.00
[ 20s ] thds: 4 tps: 104.49 qps: 2089.73 (r/w/o: 1462.81/417.95/208.97) lat (ms,95%): 51.02 err/s: 0.00 reconn/s: 0.00
[ 30s ] thds: 4 tps: 107.62 qps: 2152.04 (r/w/o: 1506.64/430.17/215.23) lat (ms,95%): 49.21 err/s: 0.00 reconn/s: 0.00
[ 40s ] thds: 4 tps: 109.86 qps: 2197.47 (r/w/o: 1538.02/439.73/219.72) lat (ms,95%): 51.02 err/s: 0.00 reconn/s: 0.00
[ 50s ] thds: 4 tps: 112.04 qps: 2240.84 (r/w/o: 1568.59/448.17/224.08) lat (ms,95%): 44.17 err/s: 0.00 reconn/s: 0.00
[ 60s ] thds: 4 tps: 111.80 qps: 2235.59 (r/w/o: 1565.19/446.80/223.60) lat (ms,95%): 45.79 err/s: 0.00 reconn/s: 0.00

SQL statistics:  # SQL 统计信息
    queries performed:
        read:                            89544  # 执行的读操作数量
        write:                           25584  # 执行的写操作数量
        other:                           12792  # 执行的其它操作数量
        total:                           127920 # 全部总数
    transactions:                        6396   (106.53 per sec.)  # 总事务数（执行事务的平均速率）
    queries:                             127920 (2130.62 per sec.) # 查询数（平均每秒能执行多少次查询）
    ignored errors:                      0      (0.00 per sec.)    # 忽略错误数
    reconnects:                          0      (0.00 per sec.)    # 重新连接次数

General statistics:  # 通用统计信息
    total time:                          60.0372s  # 测试总耗时
    total number of events:              6396      # 总请求数量(读、写、其它)

Latency (ms):  # 延迟信息
         min:                                   15.72  # 最小延迟
         avg:                                   37.53  # 平均延迟
         max:                                   324.36  # 最大延迟
         95th percentile:                       51.94  # 前 95% 的延迟
         sum:                                   240039.46  # 总延迟

Threads fairness:  # 线程稳定性
    events (avg/stddev):           1599.0000/4.74  # 事件数(平均值/标准差)
    execution time (avg/stddev):   60.0099/0.01    # 执行时间(平均值/标准差)
```

3.5 关于使用 sysbench 的一些建议

到目前为止，你应该知道了什么是 sysbench，以及怎样利用 sysbench 对数据库做基准测试。最后，我来阐述几点建议：

- 如果想要使用真实数据做基准测试，可以自行创建测试表和插入数据，不走 sysbench 的 prepare
- 基准测试要执行多次才有意思，且建议每次测试时长不低于1个小时
- 基准测试一定要模拟多线程的情况，单线程不但无法模拟真实的环境，也无法模拟阻塞以及死锁的情况

4 总结

理解和执行基准测试并没有很高的难度，但是如何判断数据库性能的高低就是一门学问了。这需要你去根据基准测试结果不断的调整数据库参数（有时候甚至需要调整操作系统的参数），毋庸置疑，这是一件麻烦且非常消耗精力的事。所以，一定不要集中化，有空的时候做做基准测试，收集并比对之前的结果，逐步调优。

5 问题

你之前做过哪方面的基准测试？是用什么工具做的？

你能说说 sysbench 工具自带的 lua 脚本是用来做哪些类型的测试吗？

使用 sysbench 自带的脚本对 MySQL 做基准测试，并分析测试结果？

6 参考资料

《高性能 MySQL（第三版）》

sysbench github

MySQL 官方文档：Measuring Performance (Benchmarking)

MySQL 官方文档：High Availability and Scalability

}

12 死锁是怎么出现的？又是怎么解决的呢？

14 你应该知道的系统数据库及常用系统表