【总结】第二章：MySQL基准测试(HP-MySQL 3rd)

2016-08-16 | 21

> High Performance MySQL, 3rd Edition

Benchmarking：就是设计一套方案来基准测试你的系统。目标是了解系统的性能、情况、行为（不止是满足并发之类的要求），在将损毁系统复制到新硬件时也会进行基准测试。这一节我们讨论压力基准测试原因、方法、策略，推荐使用sysbench工具。

基准测试的原因

一般采用假数据来基准测试，有以下几种基准测试方式：

- 提出假设、然后验证。
- 重复一下会导致系统崩溃的行为。
- 测试性能，比如并发，最好和历史数据对比，才能做出有利的改变。
- 了解数据量暴增后的表现。以后用户多了，数据增长了，提前了解升级方案。
- 测试应用在不同环境下的表现，比如高并发时、配置文件改动时。
- 测试在不同硬件、软件、操作系统下的表现（感觉好无聊）。RAID 5是否比RAID 10表现更好? 随机写入的性能是否在你从ATA磁盘切换至SAN存储时更好感觉好无聊? Linux2.4内核是否比2.6更好？MySQL升级后是否效果更好？不同引擎的效果如何？
- 测试你新购买的硬件与应用正确适配了。推荐测试新机器后才投入生产环境。

也可进行单元测试，但这里我们只关注性能。

与生产环境相比，基准测试时情况比较简单，但有利于我们得到结论。
基准测试有几个方向：能存储多大的数据、能存储什么样的数据、能进行什么样的查询，最重要的可能是速度（性能）。因为条件越苛刻（如并发）速度越慢，所以我们需要测试出各种条件阈值，来保证速度。注意，基准测试也受测试工具的影响。

预计未来所需要的性能是非常困难的。比如，不能单纯的假设用户翻了40倍，数据量就翻了40倍，那么只对40倍的数据做性能测试。因为用户多了，数据之间的连接也多了，比如表与表之间的连接查询。而且新功能也加了很多。

基准测试只供了解一些基本情况，比如了解是否还有各方面的资源还有足够多的剩余。用可以真实的数据做测试不是基准测试，这种测试的设计更为复杂，但基准测试简单、方便进行，所以基于这些情况理解基准测试数据的含义。

基准测试的策略

两种策略，全栈式：将整个应用一起测；分离式：只测MySQL。
进行全栈式的原因如下：

- 因为我们最终并不关心MySQL的性能，而是整个应用的性能。而影响整个应用性能是多方面影响的：web服务器、代码、网络、数据库。
- MySQL不一定是瓶颈。
- 可以了解各部分缓存的情况。
- 基准测试整个应用时效果较好。

但整个应用的基准测试比较难设计，设计错误的话、结论错误、决策也错误，就没啥用。

分离式的优势如下：

- 比较数据库的表与查询语句的情况。
- 已经明确了问题，看看问题发生的原因。
- 测试周期短，更能应对变化。

重复测试不同语句在数据库上的表现非常有用，另外不要直接用生产环境的数据，用它的备份数据。

尽量用真实数据，没有的话只有自己伪造。

测试目标

测试前明确目标非常有必要，其次是明确工具和技术。试着用问题来描述目标，比如：这个CPU比另一个更好吗？新的索引是否比旧的索引效果更好？

不同测试目标需要不同的方案，比如延迟和吞吐量各有各的方案。

常见指标如下：

吞吐量

吞吐量是指单位时间内完成的事务次数。对于交互性应用极其重要，数据库开发工程师为了这个指标做了很大的努力。单位时间一般是每分、每秒。

延迟或反应时间

每个任务返回的时间。单位一般是分、秒、毫秒、微妙等。一般会计算平均值、最大值、最低值、百分位值。最常用的是百分位值的反应时间，比如95%位置的反应时间是5毫秒，所以可以认为95%的任务都在5毫秒完成了。

图示化结果非常重要，比如用线表示均值和95%位值，也可以用散列图表示。下面会详谈。

并发

并发是重要而又常被误解的指标。因为网站浏览的用户不一定都在发生与web服务器甚至数据库的交互，所以并发的定义是：web服务器上同时处理的请求的数据。
不同服务器都有并发，比如web服务器的并发、MySQL的并发。web服务器并发高，肯定会带动数据库的并发，这也受到编程语言的影响。
对于MySQL，连接数和并发数是两个概念。也许web服务器和数据库之间有几百个连接，但是只有几个在执行SQL语句。

应联合吞吐量和延迟来观测并发，如果并发增加时，吞吐量降低或者延迟增加，那么应用需要改进，不然用户高峰时会有压力。
并发不是一个测量出来的值，而是需要设置的值。一般是设置一个并发值，然后观察应用、数据库的性能。当使用*sysbench*的线程数是32、64、128时，检查数据库的性能并记录下Threads_running的值。11章会解释其用意。

稳定性

在变化的环境如果如何维持性能。简单的定义就是当我们加倍资源(比如CPU)时，吞吐量也回加倍，当然延迟也是能接受的。详细定义在11章谈。性能往往不是随着资源线性增长的，一般来说会快速下降。
稳定性展现了性能可能的缺陷，比如在一个连接的情况下表现不错，但是当并发上去了之后，性能会极快的下降。
有些作业，如统计性的，只是固定的时间需要较长的时间，注意这件事情与别的事务是否冲突，它可能会导致用户交互的数据变慢。

总得来说，制定个性化的方案是必要的。比如用户接受的延迟是多少？期待的并发是多少？设计基准测试满足这些要求。

测试方案

指定方案前，我们先看看有哪些常见的错误。

- 数据集准备不充分，明知道要处理几百G的数据，缺只用几G的数据做测试。

- 数据分布不均。注意把控常见数据和不常见数据的处理。有点难理解，大概产生模拟数据的时候尽量贴近生产环境的数据分布。
- 使用不是真实状况的参数，比如假设所有个人简介都会被同等概率的查看。像微博似的，名人的微博肯定比普通人的微博浏览的次数高。
- 测试时只用一个用户，真实环境是多用户并发。
- 在单个服务器上测试分布式应用。
- 没有考虑到真实用户的行为。比如请求网页后浏览，而非马上点击下一个链接。
- 重复测试同一条SQL语句，命中缓冲，无法模拟真实状况。
- 忽略异常。比如一条平时很慢的语句突然很快速的完成了，这又异常。其次注意检查错误日志。
- 忽略机器重启后热身时间。这段时间性能较差，比如缓存没准备好。
- 使用默认配置。更多优化之后讲解。
- 测试很快结束。测试应持续一段时间。

尽量使用真实的数据，但是如果只是在新主机上测试网络状况，那用用假的无所谓。

设计测试方案

首先是明确问题和目标。其次是使用标准测试方案还是个性化设置。
选择成熟的标准测试也需要匹配目标，比如用TPC-H测试一个商业系统，但是其实其不适用OLTP系统。

个性化设置的话，过程是迭代的。其次可以从日志中获得不同时期的查询语句，作为需要优化的目标，注意覆盖不同场景，比如高峰时、批处理时。

可以从不同的日志（web、MySQL）获得查询过的语句，测试这些语句注意在不同的线程上允许，确保模拟真实状况。
对每一个连接都建立一个线程也非常重要，日志也会记录哪个连接打印了哪句查询语句。

设计好测试方案后，一般会重复多次，所以请清晰的记录过程。比如使用了哪些数据、如何配置、如何测、如何分析、如何让机器暖身。

方案可以用脚本完成，注意代码要灵活，使其可以配置不同的参数，结果也要分别保留下来；分析结果也可以用脚本来完成。

测试持续多久

一般都会关心服务器稳定时期的性能，但注意这会耗费一定资源。服务器一般会有些富余来应对短期高峰，此时应该停止测试，免得影响正常服务。
如果不知道什么时候测试，那就一直开着好了，直到清楚服务器性能。下图是一个例子，I/O的吞吐量随时间的变化：

系统热身之后，读的活动在三、四个小时内逐渐平稳，但是写持续了八个小时，并且有几次高峰。之后读写都看起来平稳了。有一个基本原则就是等到刚启动的机器表现逐渐平稳再关掉测试。我们在八小时后关闭了测试。短时间的测试结果不可信。

获得系统的性能和状态

尽可能获得更多的系统信息，便于以后得分析。将每次测试的结果存于一个目录，比如结果、配置、脚本等。常见的比如：CPU、I/O、网络流量、来自SHOW GLOBAL STATUS的常见信息。

下面一个脚本用于收集MySQL的数据：

```
#!/bin/sh
INTERVAL=5
PREFIX=$INTERVAL-sec-status RUNFILE=/home/benchmarks/running
mysql -e 'SHOW GLOBAL VARIABLES' >> mysql-variables
while test -e $RUNFILE; do
        file=$(date +%F_%I)
        sleep=$(date +%s.%N | awk "{print $INTERVAL - (\$1 % $INTERVAL)}")
        sleep $sleep
        ts="$(date +"TS %s.%N %F %T")"
        loadavg="$(uptime)"
        echo "$ts $loadavg" >> $PREFIX-${file}-status
        mysql -e 'SHOW GLOBAL STATUS' >> $PREFIX-${file}-status &
        echo "$ts $loadavg" >> $PREFIX-${file}-innodbstatus
```

```
14          mysql -e 'SHOW ENGINE INNODB STATUS\G' >> $PREFIX-${file}-innodbstatus & echo "$ts $loadavg" >> $PREFIX-${fi
15          mysql -e 'SHOW FULL PROCESSLIST\G' >> $PREFIX-${file}-processlist &
16          echo $ts
17  done
18  echo Exiting because $RUNFILE does not exist.
```

上面有些数据也许读者现在没有理解其价值，以后就会明白的。

- 时间的处理是为了最后的得到的时间能够被5除，方便以后的可视化。单纯的sleep 5的话，脚本执行的时间和系统间不同的时间戳(15:32:18.218192 和 15:32:23.819437)会干扰后续处理。
- 文件用日期和小时分割，便于查找。
- 特别的时间戳：如下所示
  TS 1471523325.003168011 2016-08-18 20:28:45 20:28:45 up 347 days, 4:56, 9 users, load average: 0.10, 0.20, 0.24

简短的代码只是提供一个思路，但也非常容易扩展，比如*/proc/diskstats*记录关于磁盘I/O的情况。

获得准确的结果

可以使用回答的问题的方式来设计测试：选择正确的指标了吗？测试的数据正确吗？选择正确的测试工具了吗？比如瓶颈在CPU，但却选了测试I/O的测试工具。

其次，确定重复测试会产生相同的结果。比如重启机器，以获得可重复的结果，但是如果重启时又加入几个随机查询，那么结果就不可能重复了。

如果测试改变了表结构或者数据，请注意重复测试时重置。插入的数据量不同也会导致不可重复测试。

确保测试不被突发事件打扰，比如长日志、*cron*的任务，网络高峰，为此请监控机器情况。

不同测试之间不要一下改变太多参数，不然不知道是什么造成的。迭代的设置参数，采用分治的思想。

有时是将Oracle的数据迁移至MySQL，想马上了解下MySQL的表现，这很麻烦，因为两者对表结构和SQL语句的处理不同，所以需要重新设计与编写才能了解真正的性能。操作系统偶尔也会影响。

默认配置也不会获得太好的影响，因为它为了小应用而设计，没有使用大量内存。

固态硬盘的出现(SSD)也影响了测试，第九章将详谈。

最后，别忽略任何一个小的异常点。因为这很有可能暗示着一个巨大的问题或缺陷。很多人怕麻烦而忽略，最终都铸成大错。

执行测试与分析结果

执行测试尽量自动化，这样可以避免忘了什么步骤，其次就是便于文档化。因为步骤不清晰，再次重复非常困难，第一次的结果也有可能没用，所以一定要自动化，尽可能的自动化。

执行次数看实际需要和情况，一般来说如果变化较大，那么执行次数、时间应响应增加。

分析的理想结果是得到结论：更新CPU使得在保持延迟的情况下增加50%的吞吐量;索引增加查询速度。更专业的做法推荐阅读统计学相关的*零假设(null hypothesis)* 的知识。

分析尽量自动化，为了也是可重复和文档化。下面的代码是从上面那份代码中提取规则化的数据。

```
1   #!/bin/sh
2   # This script converts SHOW GLOBAL STATUS into a tabulated format, one line # per sample in the input, with the metr
3   # between samples.
4   awk '
```

```
BEGIN {
        printf "#ts date time load QPS";
        fmt = " %.2f";
}
/^TS/ { # The timestamp lines begin with TS.
        ts = substr($2, 1, index($2, ".") - 1);
        load = NF - 2;
        diff = ts - prev_ts;
        prev_ts = ts;
        printf "\n%s %s %s %s", ts, $3, $4, substr($load, 1, length($load)-1);
}
/Queries/ {
        printf fmt, ($2-Queries)/diff;
        Queries=$2
}
' "$@"
```

> 以我现在的经历，推荐还是使用python。shell易读性差。这代码我可能是不会用的。

上面的代码可得到类似下面的产出：

```
[baron@ginger ~]$ ./analyze 5-sec-status-2011-03-20
#ts date time load QPS
1300642150 2011-03-20 17:29:10 0.00 0.62
1300642155 2011-03-20 17:29:15 0.00 1311.60
1300642160 2011-03-20 17:29:20 0.00 1770.60
1300642165 2011-03-20 17:29:25 0.00 1756.60
1300642170 2011-03-20 17:29:30 0.00 1752.40
1300642175 2011-03-20 17:29:35 0.00 1735.00
1300642180 2011-03-20 17:29:40 0.00 1713.00
1300642185 2011-03-20 17:29:45 0.00 1788.00
1300642190 2011-03-20 17:29:50 0.00 1596.40
```

第一行没用，因为它在测试开始前就执行了。QPS是指：每秒执行的查询（query）。这是一个基本的指标。

图的重要性

图方便我们发现问题，单纯看数、看均值容易是我们忽略问题。所以尽量使用图展示数据，如下图所示:

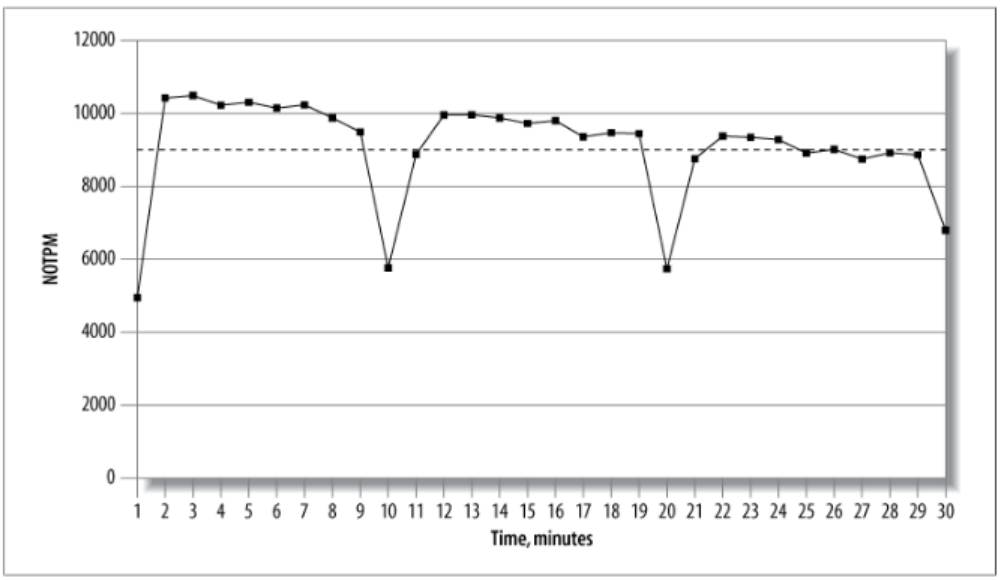

Figure 2-3. Results from a 30-minute dbt2 benchmark run

即使看95%百分位值，也会忽略掉，定期发生的*数据大量冲洗到磁盘(furious flushing)*的事件。图中Y轴表示：*NOTPM（new-order transactions per minute,每分钟事务数）*。

发生这种事件的原因很多，如旧版本的InnoDB冲洗算法不佳，进程中许多线程被卡住了，使用SHOW ENGINE INNODB STATUS、SHOW FULL PROCESS LIST来观察更多数据才能得出结论。

#测试工具

有很多现成的工具可以使用，对应测试策略，工具也分两种：全栈式、分离式。

全栈式工具

主要测试web应用：

- ab: 常用、好用的工具。它尽可能快的发出请求，测试web服务器每秒完成的请求数。
- http_load: 更为灵活的工具，可以测试一个文本内的多个URL。也可以定期发送请求。
- JMeter: 更为灵活。有可视化工具、参数配置、也可以测试FTP服务器。

分离式测试工具

- mysqlslap: MySQL的一部分，可模拟访问数据库，获得性能报告，也可指定并发连接之类的，可指定一句、多句SQL、也可自动生成SQL。
- MySQL Benchmark Suite (sql-bench): 可测试SQL语句执行的多快。作为一个套件，它包含许多成型方案，所以方便我们去比较不同引擎、配置。缺点是这个工具是单线程的，而且只能测试自己的数据。
- Super Smack: 测试 MySQL、PostgreSQL，可模拟多用户、加载测试数据近数据库，用随机数据填满数据库。
- sysbench: 最常用的，它会测试整个系统的性能，主要是对数据库重要的指标。如文件I/O的性能、OS调度、内存加载、传送速度、线程。sysbench支持lua语言，这使得sysbench可个性化设置满足不同的测试场景。

> MySQL BENCHMARK() 函数:能测测某个操作的时间，比如MD5一个字符串、SHA1一个字符串的时间。不过一般也没啥用，因为我们关心的是服务器整体的性能。

测试例子

本节我们使用不同的工具，做几种测试，供读者了解和选择。

http_load

首先准备一些url，存入urls.txt

> http://www.mysqlperformanceblog.com/
> http://www.mysqlperformanceblog.com/page/2/
> http://www.mysqlperformanceblog.com/mysql-patches/
> http://www.mysqlperformanceblog.com/mysql-performance-presentations/
> http://www.mysqlperformanceblog.com/2006/09/06/slow-query-log-analyzes-tools/

最简单的是一条一条url的取和发送，尽可能快的发送请求，也就是发送一个，返回一个，马上发送下一个。

```
$ http_load -parallel 1 -seconds 10 urls.txt
19 fetches, 1 max parallel, 837929 bytes, in 10.0003 seconds
44101.5 mean bytes/connection
1.89995 fetches/sec, 83790.7 bytes/sec
msecs/connect: 41.6647 mean, 56.156 max, 38.21 min
msecs/first-response: 320.207 mean, 508.958 max, 179.308 min
HTTP response codes:
code 200 - 19
```

下面是并发5个用户

```
$ http_load -parallel 5 -seconds 10 urls.txt
94 fetches, 5 max parallel, 4.75565e+06 bytes, in 10.0005 seconds
50592 mean bytes/connection
9.39953 fetches/sec, 475541 bytes/sec
msecs/connect: 65.1983 mean, 169.991 max, 38.189 min
msecs/first-response: 245.014 mean, 993.059 max, 99.646 min
HTTP response codes:
code 200 - 94
```

改变发送请求的频率，改为每秒发送5次：

```
$ http_load -rate 5 -seconds 10 urls.txt
48 fetches, 4 max parallel, 2.50104e+06 bytes, in 10 seconds
52105 mean bytes/connection
4.8 fetches/sec, 250104 bytes/sec
msecs/connect: 42.5931 mean, 60.462 max, 38.117 min
msecs/first-response: 246.811 mean, 546.203 max, 108.363 min
HTTP response codes:
code 200 - 48
```

增加频率，每秒20个。

```
$ http_load -rate 20 -seconds 10 urls.txt
111 fetches, 89 max parallel, 5.91142e+06 bytes, in 10.0001 seconds
53256.1 mean bytes/connection
11.0998 fetches/sec, 591134 bytes/sec
msecs/connect: 100.384 mean, 211.885 max, 38.214 min
msecs/first-response: 2163.51 mean, 7862.77 max, 933.708 min
HTTP response codes:
code 200 – 111
```

MySQL Benchmark Suite

运行这个套装需要Perl。它在MySQL的安装目录下，比如*/usr/share/mysql/sql-bench/*

运行例子：

```
cd/usr/share/mysql/sql − bench/sql − bench./run-all-tests –server=mysql –user=root –log –fast
Test finished.
You can find the result in: output/RUN-mysql_fast-Linux_2.4.18_686_smp_i686
```

注意可能耗时较长。–log 可以观察进度。下面的测试结果的例子：

```
sql-bench$ tail -5 output/select-mysql_fast-Linux_2.4.18_686_smp_i686
Time for count_distinct_group_on_key (1000:6000):
  34  wallclock secs ( 0.20 usr  0.08 sys +  0.00 cusr  0.00 csys =  0.28 CPU)
Time for count_distinct_group_on_key_parts (1000:100000):
  34  wallclock secs ( 0.57 usr  0.27 sys +  0.00 cusr  0.00 csys =  0.84 CPU)
Time for count_distinct_group (1000:100000):
  34  wallclock secs ( 0.59 usr  0.20 sys +  0.00 cusr  0.00 csys =  0.79 CPU)
Time for count_distinct_big (100:1000000):
  8   wallclock secs ( 4.22 usr  2.20 sys +  0.00 cusr  0.00 csys =  6.42 CPU)
Total time:
  868 wallclock secs (33.24 usr  9.55 sys +  0.00 cusr  0.00 csys = 42.79 CPU)
```

表明了*count_distinct_group_on_key (1000:6000)*耗费的时间，后面还有一段是测试工具耗费的时间，所以 34 - 0.28 = 33.72 才是我们关心的指标。

除了执行这个套件，也可以执行单个测试。如insert测试：

```
sql-bench$ ./test-insert
Testing server 'MySQL 4.0.13 log' at 2003-05-18 11:02:39

Testing the speed of inserting data into 1 table and do some selects on it.
The tests are done with a table that has 100000 rows.

Generating random keys
Creating tables
Inserting 100000 rows in order
Inserting 100000 rows in reverse order
Inserting 100000 rows in random order
Time for insert (300000):
  42 wallclock secs ( 7.91 usr  5.03 sys +  0.00 cusr  0.00 csys = 12.94 CPU)
Testing insert of duplicates
Time for insert_duplicates (100000):
  16 wallclock secs ( 2.28 usr  1.89 sys +  0.00 cusr  0.00 csys =  4.17 CPU)
```

sysbench

sysbench可进行各种各样的测试，不仅是数据库的性能，还包含一个系统作为数据库服务器的性能。强烈推荐熟悉sysbench，尽可能使用它测试。

CPU测试

使用下面的语句完成CPU测试，结果非常简单，就是完成的秒数，–cpu-max-prime是计算小于这个数的最大的素数。

```
sysbench –test=cpu –cpu-max-prime=20000 run
```

I/O测试

sysbench会模拟InnoDB的行为完成I/O测试，这也为什么我们要使用sysbench的原因。
首先我们要准备文件，文件必须足够大，内存无法全部装载而必须进行磁盘的读入动作。

```
sysbench –test=fileio –file-total-size=150G prepare
```

当前目录会生成一个大文件供读写，下面几个参数用于测试不同的I/O操作：

- seqwr: Sequential write
- seqrewr: Sequential rewrite
- seqrd: Sequential read

- rndrd: Random read
- rndwr: Random write
- rndrw: Combined random read/write

下面的语句进行随机I/O测试：

```
$ sysbench –test=fileio –file-total-size=150G –file-test-mode=rndrw/ –init-rng=on –max-time=300 –max-requests=0 run
```

结果：

```
sysbench v0.4.8: multithreaded system evaluation benchmark
Running the test with following options:
Number of threads: 1
Initializing random number generator from timer.

Extra file open flags: 0
128 files, 1.1719Gb each
150Gb total file size
Block size 16Kb
Number of random requests for random IO: 10000
Read/Write ratio for combined random IO test: 1.50
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing random r/w test
Threads started!
Time limit exceeded, exiting…
Done.

Operations performed: 40260 Read, 26840 Write, 85785 Other = 152885 Total
Read 629.06Mb Written 419.38Mb
Total transferred 1.0239Gb (3.4948Mb/sec) 223.67 Requests/sec executed

Test execution summary: 300.0004s
total time: 67100
total number of events: 254.4601
total time taken by event execution: per-request statistics:
min: 0.0000s
avg: 0.0038s
max: 0.5628s
approx. 95 percentile: 0.0099s
Threads fairness:
events (avg/stddev): 67100.0000/0.00
execution time (avg/stddev): 254.4601/0.00
```

上面最终的信息就是每秒请求数:223.67 Requests/sec 和吞吐量 1.0239Gb。95%值也透露了一些信息。这些都标志着性能的好坏。

测试完成后，执行语句销毁文件：

```
$ sysbench –test=fileio –file-total-size=150G cleanup
```

OLTP测试

下面的例子是对一个有百万行的表测试OLTP，首先是准备表：

> **$ sysbench –test=oltp –oltp-table-size=1000000 –mysql-db=test/ –mysql-user=root prepare**
> sysbench v0.4.8: multithreaded system evaluation benchmark
> No DB drivers specified, using mysql
> Creating table ‘sbtest’…
> Creating 1000000 records in table ‘sbtest’…

下面是用8个并发线程，持续60秒的读模式：

```
$ sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test --mysql-user=root/
--max-time=60 --oltp-read-only=on --max-requests=0 --num-threads=8 run
sysbench v0.4.8:  multithreaded system evaluation benchmark

No DB drivers specified, using mysql
WARNING: Preparing of "BEGIN" is unsupported, using emulation
(last message repeated 7 times)
Running the test with following options:
Number of threads: 8

Doing OLTP test.
Running mixed OLTP test
Doing read-only test
Using Special distribution (12 iterations,  1 pct of values are returned in 75 pct
cases)
Using "BEGIN" for starting transactions
Using auto_inc on the id column
Threads started!
Time limit exceeded, exiting...
```

```
(last message repeated 7 times)
Done.

OLTP test statistics:
    queries performed:
        read:                            179606
        write:                           0
        other:                           25658
        total:                           205264
    transactions:                        12829  (213.07 per sec.)
    deadlocks:                           0      (0.00 per sec.)
    read/write requests:                 179606 (2982.92 per sec.)
    other operations:                    25658  (426.13 per sec.)

Test execution summary:
    total time:                          60.2114s
    total number of events:              12829
    total time taken by event execution: 480.2086

    per-request statistics:
         min:                            0.0030s
         avg:                            0.0374s
         max:                            1.9106s
         approx. 95 percentile:          0.1163s

Threads fairness:
    events (avg/stddev):           1603.6250/70.66
    execution time (avg/stddev):   60.0261/0.06
```

信息很多，重要的有：

1. 事务数量
2. 每秒事务数
3. 平均每个事务的时间的统计。
4. 线程相关的统计

最新版本的sysbench支持多张表联合测试，你可以在一定间隔内观测吞吐量和反应时间。这些指标对理解系统行为非常重要。

sys其他特定

它可以测试一些常见的性能指标

- memory: 发起连续的内存读写。
- 线程：测试线程调度的性能。
- 互斥锁：模拟一个高并发的情景，测试互斥锁的性能。
- 连续读：测试连续读，这是非常重要的。这个指标可以向你展示RAID控制器的缓存性能并提醒你结果问题。比如当在没有电池备份的读缓存（no battery-backed write cache）的情况下，达到了每秒达到了3,000个请求，那么哪里有问题。

示例只是一小部分，更多内容参考文档。

dbt2 TPC-C on the Database Test Suite

(略)，急着学后面几章。

Percona's TPCC-MySQL Tool

(略)，急着学后面几章。

总结

基准测试是很有教育意义的，从描述一个问题到用测试结果来回答它，与数学课上的用公式描述文字类似。解析问题、设计测试方案回答问题、选择测试的时长和参数、运行测试、收集数据、分析结果，使得MySQL表现的更好。

没用做过基准测试的话，最好体验一下*sysbench*，使用下OLTP和fileIO的测试。OLTP可能在比较不同系统时非常棘手，但是文件系统和磁盘测试，是能够快速协助快速定位或者排除问题的。我们无数次遇到这样的情况，就是尽管管理系统声明了，但是SAN还是发生了磁盘失败，RAID控制器缓存并没有正确的识别配置。（真一段真是好难理解）还有当你对单个磁盘测试时，虽然它声明了每秒执行了14000个随机读，那么这肯定是有一些错误或者配置出错。

尽快能熟悉几个需要的工具、用脚本将任务自动化起来、将数据可视化的分析，这样你的眼光将会逐渐变得敏锐起来。
(完)

#MySQL

❮用SQL求得7日均和周均值 【总结】第三章：探索（profiling）服务器表现(HP-MySQL 3rd)❯