TiDB Sysbench 测试指南

一、Sysbench 介绍

Sysbench 是一个模块化的、跨平台、开源的多线程基准测试工具，可以执行 CPU、内存、线程、IO、数据库等方面的性能测试，主要用于评估测试各种不同系统参数下的数据库负载情况，数据库目前支持 MySQL/Oracle/PostgreSQL。

二、TiDB 部署

参考 TiDB 部署文档 部署 TiDB 集群。在 3 台服务器的条件下，建议每台机器部署 1 个 TiDB, 1 个 PD, 1 个 TiKV 实例。关于磁盘，以 32 张表、每张表 10M 行数据为例，建议 TiKV 的数据目录所在的磁盘空间大于 512 GB。

对于单个 TiDB 的并发连接数，建议控制在 500 以内，如需增加整个系统的并发压力，可以增加 TiDB 实例，具体增加的 TiDB 个数视测试压力而定。

对于写密集型测试，比如 Sysbench 的 oltp_update_index、oltp_update_non_index、oltp_write_only，建议增加 TiKV 实例数量，如每台机器 2 个 TiKV。

对于机器硬件要求，请参考文档。

	TiDB	TiKV	PD
node1	1	1(2)	1
node2	1	1(2)	1
node3	1	1(2)	1

括号内为写密集测试的部署建议。

三、TiDB 集群调优

3.1 TiDB 调优

升高日志级别，可以减少打印日志数量，对性能有积极影响。开启 TiDB 配置中的 prepared plan cache，以减少优化执行计划的开销。具体在 TiDB 配置文件中加入：

```
[log]
level = "error"
[prepared-plan-cache]
```

```
enabled = true
```

如果使用的是 master 的 TiDB 版本，全局变量 tidb_max_chunk_size 是一个有负载倾向的参数，对于 OLTP 测试建议调整到 4 以下。

```
SET GLOBAL tidb_max_chunk_size = 4;
```

3.2 TiKV 调优

升高 TiKV 的日志级别同样有利于性能表现。

由于 TiKV 是以集群形式部署，在 Raft 算法的作用下，能保证大多数节点已经写入数据。因此，除了对数据安全极端敏感的场景之外，raftstore 中的 sync-log 选项可以关闭。

TiKV 集群存在两个 Column Family（Default CF 和 Write CF）主要用于存储不同类型的数据。对于 Sysbench 测试，导入的数据的 Column Family 在 TiDB 集群中的比例是固定的。这个比例是：

Default CF : Write CF = 4 : 1

在 TiKV 中需要根据机器内存大小配置 RocksDB 的 block cache，以充分利用内存。以 20 GB 内存的虚拟机部署一个 TiKV 为例，其 block cache 建议配置如下:

```
log-level = "error"
[raftstore]
sync-log = false
[rocksdb.defaultcf]
block-cache-size = "12GB"
[rocksdb.writecf]
block-cache-size = "3GB"
```

更详细的 TiKV 参数调优请参考 TiKV 性能参数调优 。

四、测试准备工作

4.1 Sysbench 配置

建议使用 Sysbench 1.0 之后的版本来进行测试，可以在这里进行下载。以下为 Sysbench 配置文件样例：

```
mysql-host={TIDB_HOST}
mysql-port=4000
mysql-user=root
mysql-db=sbtest
time=600
threads={8, 16, 32, 64, 128, 256}
```

```
report-interval=10
db-driver=mysql
```

可根据实际需求调整其参数，其中 TIDB_HOST 为 TiDB server 的 IP 地址，threads 为测试中的并发连接数，可在 “8, 16, 32, 64, 128, 256” 中调整，在导入数据时，建议设置 threads = 8 或者 16。调整后，将该文件保存为名为 config 的文件。

4.2 数据导入

MySQL 客户端执行以下 SQL，创建数据库 sbtest，

```
create database sbtest;
```

调整 Sysbench 脚本创建索引的顺序。Sysbench 导入数据是按照“建表->插入数据->创建索引”的顺序，该方式对于 TiDB 需要花费更多的导入时间。用户可以通过调整这个顺序来加速数据的导入。假设用户使用的 Sysbench 版本是 https://github.com/akopytov/sysbench/tree/1.0.14。我们可以通过以下两种方式来修改。

1. 直接下载为 TiDB 修改好的 oltp_common.lua 文件，覆盖 /usr/share/sysbench/oltp_common.lua 文件。
2. 将 /usr/share/sysbench/oltp_common.lua 的 235 ～ 240 行移动到 198 行以后。

此操作为可选操作，仅节约了导入数据的时间。

命令行键入以下命令，开始导入数据，

```
sysbench --config-file=config oltp_point_select --tables=32 --table-
size=10000000 prepare
```

4.3 数据预热与统计信息收集

数据预热可将磁盘中的数据载入内存的 block cache 中，预热后的数据对系统整体的性能有较大的改善，建议在每次重启集群后进行一次预热。

Sysbench 没有提供数据预热的功能，因此需要手动进行数据预热。以 Sysbench 中某张表 sbtest1 为例，执行如下 SQL，可实现数据预热效果：

```
SELECT COUNT(pad) FROM sbtest1 USE INDEX (k_1);
```

统计信息收集有助于优化器选择更为准确的执行计划，可以通过

```
ANALYZE TABLE sbtest1;
```

来收集表 sbtest1 的统计信息，其他表方式类似。

五、只读测试

点查命令：

```
sysbench --config-file=config oltp_point_select --tables=32 --table-
size=10000000 run
```

综合只读查询（包含范围查询、sum、order by 和 distinct）命令：

```
sysbench --config-file=config oltp_read_only --tables=32 --table-
size=10000000 run
```

六、写入测试

UPDATE 测试是很好的写入测试，对 TiKV 压力较大，如发现性能不如预期，建议增加 TiKV 实例。

更新带索引列命令：

```
sysbench --config-file=config oltp_update_index --tables=32 --table-
size=10000000 run
```

更新不带索引列命令：

```
sysbench --config-file=config oltp_update_non_index --tables=32 --table-
size=10000000 run
```

纯写测试包含了 INSERT/DELETE/UPDATE，对于单纯的 INSERT 或 DELETE 测试，由于测试集群的配置、测试时间等因素不同的关系，难以给出一个具有参考意义的测试评估。如果希望测试 INSERT/DELETE 性能，建议也通过测试 oltp_write_only 来进行测试 。命令如下：

```
sysbench --config-file=config oltp_write_only --tables=32 --table-
size=10000000 run
```

七、读写复合测试

包含了以上所有测试的全部语句。命令如下：

```
sysbench --config-file=config oltp_read_write --tables=32 --table-
size=10000000 run
```

八、Benchmark 参考

我们选择客户常用的阿里云作为验证 Sysbench 性能的标准平台，购买了以下主机，并部署 TiDB 集群。

名称	操作系统	内网 IP	CPU	内存	实例规格	镜像 ID

sysbench-tikv-1	CentOS 7.4 64 位	192.168.20.12	8	65536	ecs.i2.2xlarge	centos_7_04_64_20G_alibase_201701015.vhd
sysbench-tikv-2	CentOS 7.4 64 位	192.168.20.11	8	65536	ecs.i2.2xlarge	centos_7_04_64_20G_alibase_201701015.vhd
sysbench-tikv-3	CentOS 7.4 64 位	192.168.20.13	8	65536	ecs.i2.2xlarge	centos_7_04_64_20G_alibase_201701015.vhd
sysbench-tidb-1	CentOS 7.4 64 位	192.168.20.9	12	24576	ecs.c5.3xlarge	centos_7_04_64_20G_alibase_201701015.vhd
sysbench-tidb-2	CentOS 7.4 64 位	192.168.20.8	12	24576	ecs.c5.3xlarge	centos_7_04_64_20G_alibase_201701015.vhd
sysbench-tidb-3	CentOS 7.4 64 位	192.168.20.10	12	24576	ecs.c5.3xlarge	centos_7_04_64_20G_alibase_201701015.vhd
sysbench-pd-1	CentOS 7.4 64 位	192.168.20.7	8	16384	ecs.c5.2xlarge	centos_7_04_64_20G_alibase_201701015.vhd
sysbench-pd-2	CentOS 7.4 64 位	192.168.20.6	8	16384	ecs.c5.2xlarge	centos_7_04_64_20G_alibase_201701015.vhd
sysbench-pd-3	CentOS 7.4 64 位	192.168.20.5	8	16384	ecs.c5.2xlarge	centos_7_04_64_20G_alibase_201701015.vhd
sysbench-test-monitor	CentOS 7.4 64 位	192.168.20.4	4	16384	ecs.g5.xlarge	centos_7_04_64_20G_alibase_201701015.vhd

\\

测试了数据 32 表，每表 10M 数据。实测性能如下，

point_select	thread	TPS	QPS	avg. latency(ms)	.95 latency(ms)	max. Latency(ms)
point_select	3*8	35278.5	35278.5	0.68	0.91	33.42
point_select	3*16	56639.2	56639.2	0.85	1.58	35.60
point_select	3*32	71274.9	71274.9	1.35	3.33	38.67
point_select	3*64	78806.4	78806.4	2.44	7.39	37.60
point_select	3*128	81945.8	81945.8	4.69	12.91	80.10
read_only	thread	TPS	QPS	avg. latency(ms)	.95 latency(ms)	max. Latency(ms)
read_only	3*8	1599.93	25598.76	15.01	18.51	79.47
read_only	3*16	2312.87	37005.8	20.77	26.85	87.42
read_only	3*32	2798.28	44772.6	34.33	48.06	109.02
read_only	3*64	3039.77	48636.5	63.20	91.35	151.47
read_only	3*128	3255.78	52092.5	117.99	161.53	254.56
update_index	thread	TPS	QPS	avg. latency(ms)	.95 latency(ms)	max. Latency(ms)
update_index	3*8	3694.18	3694.18	6.50	8.53	164.18
update_index	3*16	5247.11	5247.11	9.15	12.90	170.32
update_index	3*32	6758.95	6758.95	14.20	21.63	3463.77
update_index	3*64	7657.52	7657.52	25.07	42.61	4886.06
update_index	3*128	8755.74	8755.74	43.85	80.03	5019.62
update_non_index	thread	TPS	QPS	avg. latency(ms)	.95 latency(ms)	max. Latency(ms)
update_non_index	3*8	5277.84	5277.84	4.55	6.03	2208.73
update_non_index	3*16	9929.81	9929.81	4.83	5.88	218.03
update_non_index	3*32	15173.1	15173.12	6.33	8.58	231.64
update_non_index	3*64	21149.8	21149.76	9.07	18.31	275.79
update_non_index	3*128	26898.8	26898.79	14.27	80.54	332.35
write_only	thread	TPS	QPS	avg. latency(ms)	.95 latency(ms)	max. Latency(ms)
write_only	3*8	1931.98	11591.88	12.42	16.91	4820.59
write_only	3*16	2664.62	15987.7	18.01	25.28	5019.23
write_only	3*32	2964.8	17788.76	32.37	48.92	5341.08
write_only	3*64	3981.32	23887.91	48.21	72.70	8868.65
write_only	3*128	4111.36	24668.17	93.26	145.87	9457.49
read_write	thread	TPS	QPS	avg. latency(ms)	.95 latency(ms)	max. Latency(ms)
read_write	3*8	629.28	12585.62	38.14	50.41	220.44
read_write	3*16	854.95	17098.85	56.14	76.28	348.28
read_write	3*32	1100.38	22007.45	87.24	123.29	4303.48
read_write	3*64	1278.25	25565.04	150.20	207.84	3378.02
read_write	3*128	1458.04	29160.67	263.33	383.37	5165.39

九、常见问题

1. 在高并发压力下，为什么 TiKV 的 CPU 利用率依然很低？

   TiKV 虽然整体 CPU 偏低，但部分模块可能 CPU 已经达到了很高的利用率。

   TiKV 的 raft store、async apply 和 scheduler 三个模块为单线程模块，其最多只能使用一个 CPU core。TiKV 的其他模块，如 storage readpool、coprocessor 和 grpc 的最大并发度限制是可以通过 TiKV 的配置文件进行调整的。

   通过 Grafana 的 TiKV Thread CPU 监控面板可以观察到其实际使用率。

   - 如出现单线程模块瓶颈，可以通过扩展 TiKV 节点来进行负载均摊；
   - 如出现多线程模块瓶颈，可以通过增加该模块并发度进行调整。

2. 在高并发压力下，TiKV 也未达到 CPU 使用瓶颈，为什么 TiDB 的 CPU 利用率依然很低？
   TiDB 的 grpc-concurrency 是用来控制 TiDB 一侧向 TiKV 发送 grpc 请求的并发度。若出现网络消息挤压现象，可以通过调大这个参数来解决。
   在某些高端设备上，使用的是 NUMA 架构的 CPU，跨 CPU 访问远端内存将极大损耗降低性能。TiDB 默认将使用服务器所有 CPU，goroutine 的调度不可避免的会出现跨 CPU 内存访问。因此，建议在 NUMA 架构服务器上，部署 n 个 TiDB（n = NUMA CPU 的个数），同时将 TiDB 的 max-procs 设置为一个 NUMA CPU 的核数。

3. 在高并发压力下，TiDB、TiKV 的配置都合理，为什么整体性能还是偏低？
   出现这种问题，多数情况下可能与使用了 proxy 有关。可以尝试直接对单个 TiDB 加压，将结果加和与使用 proxy 的进行对比。以 haproxy 为例，nbproc 参数可以增加其最大启动的进程数，较新版本的 haproxy 还支持 nbthread 和 cpu-map 等。都可以降低其对性能的不利影响。