测试报告——通过冷热分离减少写放大

来逸瑞，朱猛猛

一、单元测试

1. 修改单元测试

我们的改动主要集中在 blob format、blob gc picker 和 blob gc job 这 3 个模块中，因此我们在这些模块对应的测试文件中增加或修改了一些测试，用于覆盖我们的代码。

A．Blob_format_test 的修改

修改 TEST(BlobFormatTest, BlobRecord)

增加调用了 BlobRecord 的 AddCount()函数，并检查 BlobRecord 编码解码是否正确，对修改后 的 BlobRecord 进行单元测试。

修改 TEST(BlobFormatTest, BlobFileMeta)

增加了创建存储冷数据的 BlobFileMeta 的测试，并检查其编码解码是否正确，对修改后的 BlobFileMeta 进行单元测试。

B．Blob_gc_picker_test 的修改

在 AddBlobFile()函数中增加了一个参数以确定增加的文件是冷文件还是热文件。

我们在原来的基础上对存放冷数据的文件进行的 gc_picker 的一系列测试，以测试我们对冷数据的 gc_pick。

增加 TEST_F(BlobGCPickerTest, Basic_cold)

增加 TEST_F(BlobGCPickerTest, BeingGC_cold)

增加 TEST_F(BlobGCPickerTest, TriggerNext_cold)

增加 TEST_F(BlobGCPickerTest, PickFileAndTriggerNext_cold)

增加 TEST_F(BlobGCPickerTest, ParallelPickGC_cold)

对存放冷数据的文件特有的 picker 策略进行单元测试。

增加 TEST_F(BlobGCPickerTest, NoPickGC_cold)，测试一次 Gc 的冷文件大小过小时，不进行对这些冷文件进行 Gc。

增加 TEST_F(BlobGCPickerTest, Small_PickGC_cold)，测试一次 Gc 的冷文件大小达到 8MB 时，对这些冷文件进行 Gc。

对冷热文件混合的情况做测试。

增加 TEST_F(BlobGCPickerTest, PickGC_hot_and_cold)，测试一次 Gc 的冷热文件都存在时，是否正常 Gc。

C．Blob_gc_job_test 的修改

在 RunGC (…)上进行修改，产生了 RunGC_cold(…)函数。

我们希望能在执行多次 RunGc(…)后，能够产生冷数据，结果发现测试文件里的 RunGC (…)无法完成，因为 RunGc(…)不对完成 GC 后的 blob_gc 调用 ReleaseGcFiles()函数，导致 blob_gc 里的文件的 BlobFileMeta 里的 state_没有变为 FileState::kNormal，使其无法继续参加 Gc 选举，因此无法产生冷数据。

我们在 RunGC_cold(…)中对完成 GC 后的 blob_gc 调用 ReleaseGcFiles()函数。

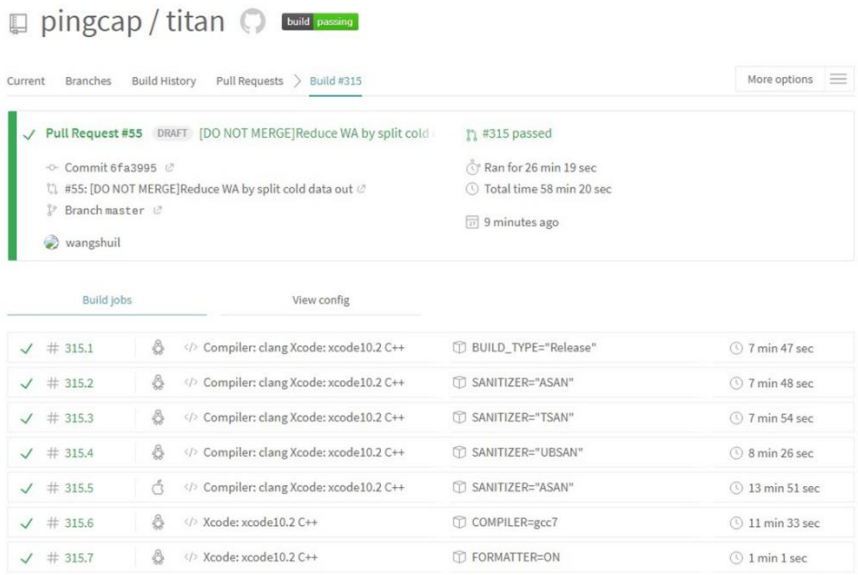
增加 TEST_F(BlobGCJobTest, Cold_Blob_Gc)测试

我们通过多次调用 RunGC_cold(…) 重写数据，来产生冷数据与冷文件。

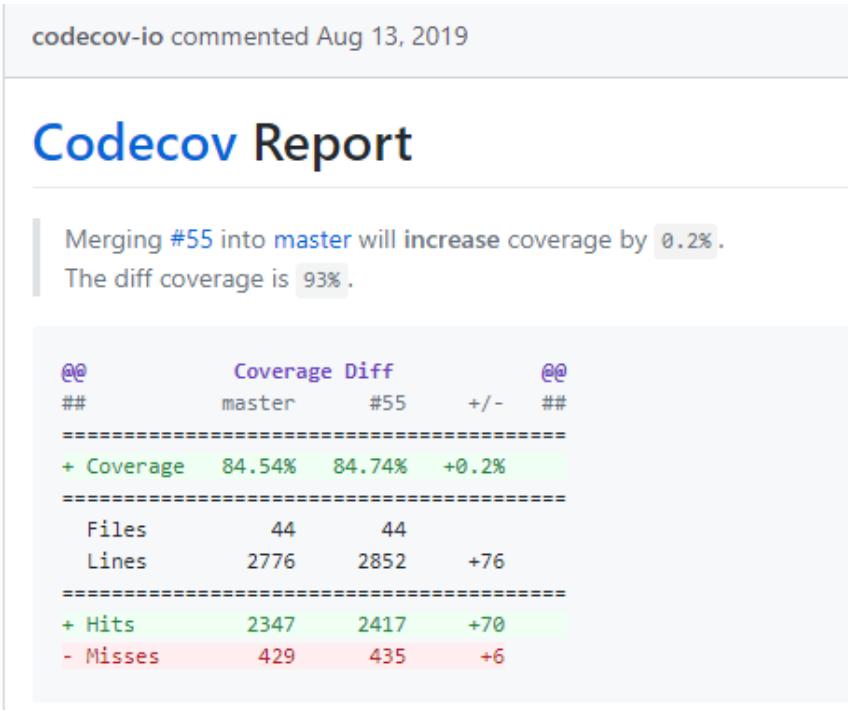
2. 运行单元测试

我们使用 Github 上的 travis-ci 工具来运行单元测试，使用 codecov 工具来计算测

试覆盖率。通过 travis-ci，我们找到并修订了一些存在内存泄漏的代码，最后通过了测试。测试结果如下图：



测试覆盖率如下图：



二、性能测试

1. 环境搭建

   我们使用 Tidb-ansible 部署了一个单节点的 TiKV 集群，节点上线后可使用 Grafana 实时监控。具体的部署方法见 https://pingcap.com/docs-cn/v3.0/how-to/deploy/orchestrated/ansible/。

2. YCSB 和 Workload

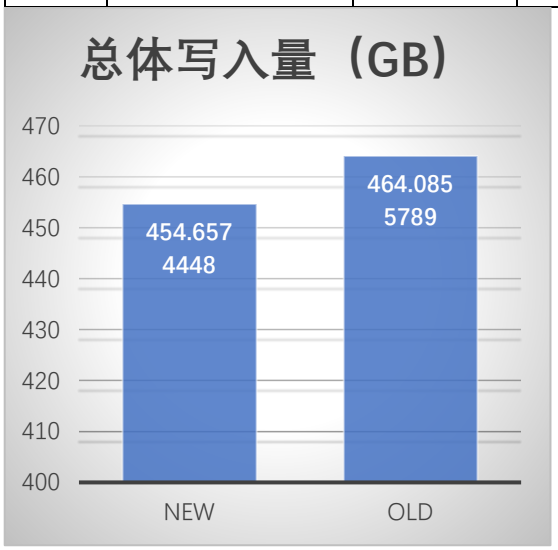
   我们使用了 pingcap 的 go-ycsb，安装运行后设计了用于我们测试的，更新写入为主导的 workload，具体的 workload 参数如下：

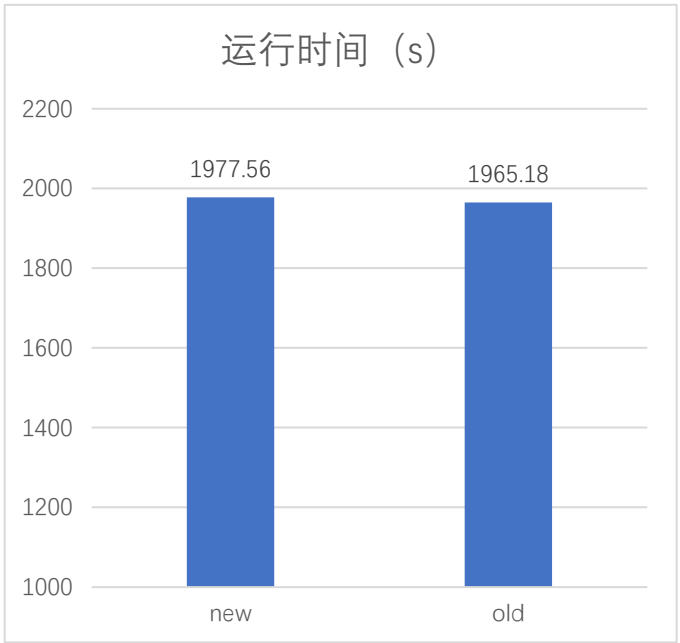
Record count : 2,000,000
value size : 8192
Update ops : 6,000,000
Insert ops : 1,000,000
Read-modify ops : 3,000,000
Hot spot data fraction=0.2
Hot spot op fraction=0.8

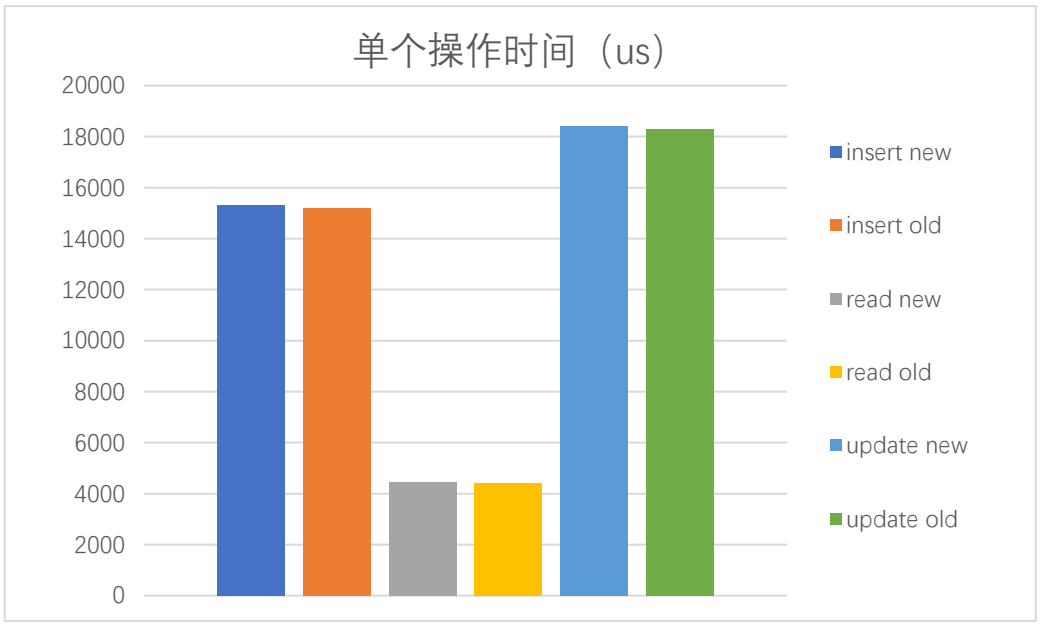
3. 运行方式和结果

我们使用 go-ycsb 依次 load 和 run 该 workload。每次 run 结束后删除数据库内容，再进行下一次测试。我们对我们的代码和原本的 Titan 都运行了 5 次该 workload，其平均结果如下：

	Total write/GB	Run time/s	Insert/us	Read/us	Update/us
New	454.6574448	1977.56	15302.6	4456	18404.8
Old	464.0855789	1965.18	15188	4424.6	18288.6






单个操作时间（us）
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
insert new
insert old
read new
read old
update new
update old