参考资料：
https://en.wikipedia.org/wiki/Profile-guided_optimization
https://doc.rust-lang.org/rustc/profile-guided-optimization.html

PGO具有一定优化效果的原因之一是TiKV-Server在运行期间有严重的iTLB-Cache-Miss现象（elf中的TEXT段比较大，30M左右）。TiDB-Server也存在同样的问题（更严重一些），但由于目前Golang编译器不支持PGO优化，所以当前TiDB-Server无法使用PGO。理论上PGO/LTO等优化方法同样适用于TiFlash。

```
^C
 Performance counter stats for process id '451919':

        448760.61 msec task-clock                #    7.879 CPUs utilized
         12166261      context-switches          #    0.027 M/sec
          2218012      cpu-migrations            #    0.005 M/sec
           472371      page-faults               #    0.001 M/sec
    1268494342888      cycles                    #    2.827 GHz                      (38.51%)
     737296495089      instructions              #    0.58  insn per cycle           (46.23%)
     137610215118      branches                  #  306.645 M/sec                    (46.17%)
       6346687936      branch-misses             #    4.61% of all branches          (46.20%)
     228936583589      L1-dcache-loads           #  510.153 M/sec                    (46.16%)
      23890653366      L1-dcache-load-misses     #   10.44% of all L1-dcache hits    (46.14%)
       5296318340      LLC-loads                 #   11.802 M/sec                    (30.68%)
        397583073      LLC-load-misses           #    7.51% of all LL-cache hits     (30.70%)
   <not supported>      L1-icache-loads
      60633380770      L1-icache-load-misses                                         (30.74%)
     230200405316      dTLB-loads                #  512.969 M/sec                    (30.82%)
       1731712817      dTLB-load-misses          #    0.75% of all dTLB cache hits   (30.84%)
       2189054185      iTLB-loads                #    4.878 M/sec                    (30.80%)
       1333416784      iTLB-load-misses          #   60.91% of all iTLB cache hits   (30.79%)
   <not supported>      L1-dcache-prefetches
   <not supported>      L1-dcache-prefetch-misses

      56.953296497 seconds time elapsed
```

TiKV-Server Perf Stat

```
 Performance counter stats for process id '10132':

        680558.81 msec task-clock                #   21.221 CPUs utilized
           790943      context-switches          #    0.001 M/sec
            38920      cpu-migrations            #    0.057 K/sec
            11874      page-faults               #    0.017 K/sec
    2017017112602      cycles                    #    2.964 GHz                      (30.76%)
    1406430854248      instructions              #    0.70  insn per cycle           (38.47%)
     272733697678      branches                  #  400.750 M/sec                    (38.46%)
       9224156383      branch-misses             #    3.38% of all branches          (38.46%)
     378467268082      L1-dcache-loads           #  556.113 M/sec                    (38.47%)
      25702325044      L1-dcache-load-misses     #    6.79% of all L1-dcache hits    (38.46%)
       6434961362      LLC-loads                 #    9.455 M/sec                    (30.76%)
       1006236409      LLC-load-misses           #   15.64% of all LL-cache hits     (30.76%)
   <not supported>      L1-icache-loads
      59798936928      L1-icache-load-misses                                         (30.76%)
     378785890952      dTLB-loads                #  556.581 M/sec                    (30.78%)
       2312187735      dTLB-load-misses          #    0.61% of all dTLB cache hits   (30.77%)
       1560219553      iTLB-loads                #    2.293 M/sec                    (30.78%)
       1242308896      iTLB-load-misses          #   79.62% of all iTLB cache hits   (30.77%)
   <not supported>      L1-dcache-prefetches
   <not supported>      L1-dcache-prefetch-misses

      32.070733246 seconds time elapsed
```

TiDB-Server Perf Stat

某混合负载下测试结果显示，TiKV-Server[PGO]优化会带来一些性能提升：集群TPS提升约3.0%，整体延迟降低约3.4%，TiKV-Server CPU节省约7.1%，TiKV-Server CPU IPC(inst per cycle)指标提升约6.6%。除了PGO [LTO]以及将elf中的TEXT段单独使用HugePage加载等方法（此方法针对TiDB-Server同样适用）均值得继续探索，另外，有第三方测试结果显示针对[内核使用LTO]优化对IO和Context Switch有比较显著的性能提升：用户空间的应用leveldb吞吐提升了8.4%，延迟降低5.9%（猜测此处吞吐与延迟指标的改善同样至少部分适用于TiKV），Context Switch速度变为原版的2.85倍。

下面描述如何给TiKV-Server开启PGO优化（以TiKV 5.1.0为例）。

1. 完成编译TiKV-Server前的准备工作
2. 构建拥有运行时PGO Profiler的TiKV-Server Binary文件，后面称之为tikv-server-with-PGO-profiler。构建过程中会需要比较多的内存资源。

   2.1. 将scripts/run-cargo.sh的第78行更改成如下：

   ```bash
   RUSTFLAGS="-Cprofile-generate=/tmp/tikv-pgo-data" cargo $args $packages --target=x86_64-unknown-linux-gnu  --features="$features" $X_CARGO_ARGS
   ```

   注：如果是arm64架构，target需要改成aarch64-unknown-linux-gnu

   2.2. 编译：
   ```bash
   ROCKSDB_SYS_STATIC=1  make dist_release
   ```

3. 将tikv-server-with-PGO-profiler作为一个TiKV实例的二进制映像运行在一个集群中，对整体集群施加负载X持续性压测，压测完毕时将上述TiKV进程退出，tikv-server-with-PGO-profiler会将运行时的profile数据dump到`/tmp/tikv-pgo-data`目录下，然后使用工具llvm-profdata对profile文件进行二次处理。

   3.1. 安装工具llvm-profdata

   ```bash
   rustup component add llvm-tools-preview
   ```

   定位绝对路径（一般在`~/.rustup/toolchains/<toolchain>/lib/rustlib/<target-triple>/bin/`）

   3.2. llvm-profdata merge

```bash
$mypath/llvm-profdata merge -o /tmp/tikv-pgo-data/merged.profdata /tmp/tikv-pgo-data
```

4. 编译最终PGO优化版本的TiKV，简称tikv-server-PGO。

4.1. 将scripts/run-cargo.sh的第78行更改成如下：

```bash
RUSTFLAGS="-Cprofile-use=/tmp/tikv-pgo-data/merged.profdata" cargo $args
$packages --target=x86_64-unknown-linux-gnu  --features="$features"
$X_CARGO_ARGS
```

注：如果是arm64架构，target需要改成aarch64-unknown-linux-gnu

4.2. 编译：

```bash
ROCKSDB_SYS_STATIC=1  make dist_release
```

最终得到针对负载X场景PGO优化版本的tikv-server-PGO。