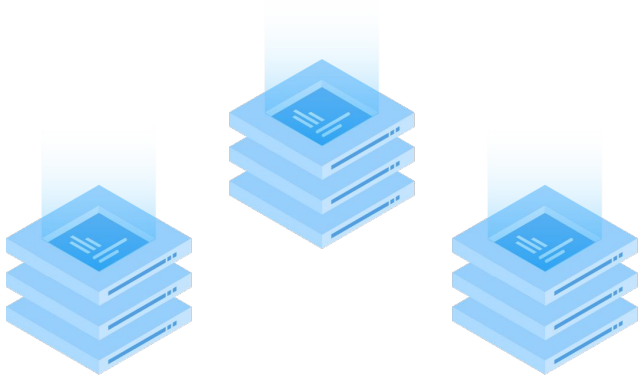
Finding Bottlenecks

Presented by Jinpeng Zhang

PingCAP TiDB



Before we begin

- Context
- Goal
- Outline
  - Performance & Bottleneck 漫谈
  - Profile
    - CPU Profile
      - Profile TiDB CPU
      - Profile TiKV CPU
    - IO Profile
      - Tools: iostat/iosnoop/fio
    - Memory Profile
      - 采用 go pprof heap 工具剖析 TiDB 内存使用
      - 三种 profile TiKV 内存的方法
    - 一个好用的 Profile 工具：VTune
  - Homework

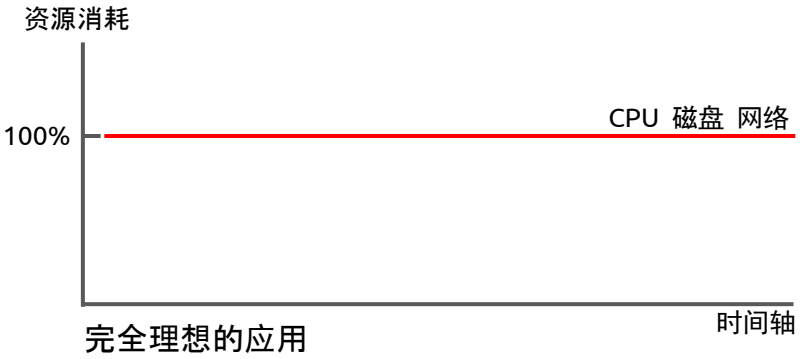
Performance & Bottleneck 漫谈

如何衡量性能

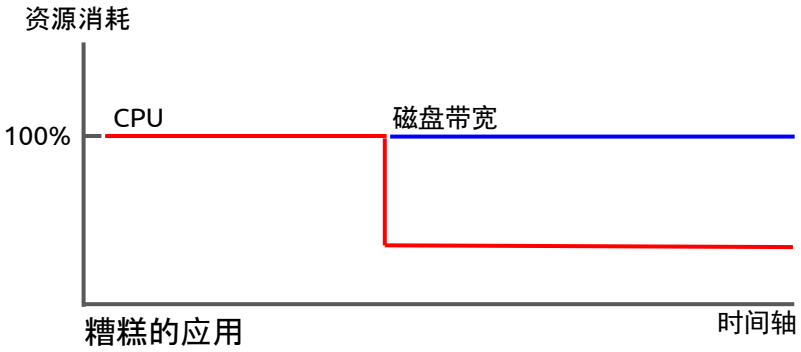
性能 = 产出 / 资源消耗

- 产出：
  - 事务次数（QPS）
  - 吞吐数据量（Bytes）
- 消耗资源：
  - CPU 时间片
  - 磁盘、网络 IO 次数、流量
- 在资源固定的情况下，提高 产出：
  - 每秒事务：QPS
    - 如果独立作为产出的衡量单位，不够准确
    - 可以通过 batch 的方式，在 QPS 不变的情况下，提升系统产出
  - 每秒吞吐：Bytes / s

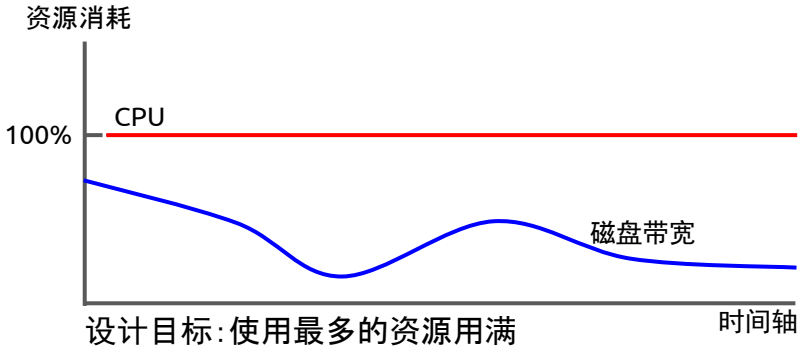
时间轴与对应的资源消耗



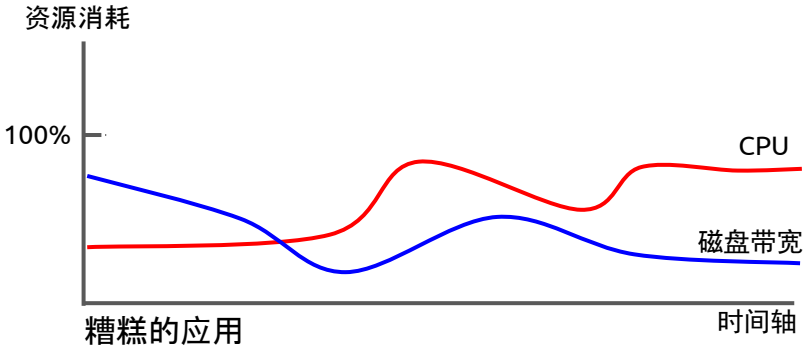
完全理想的应用



糟糕的应用



设计目标:使用最多的资源用满



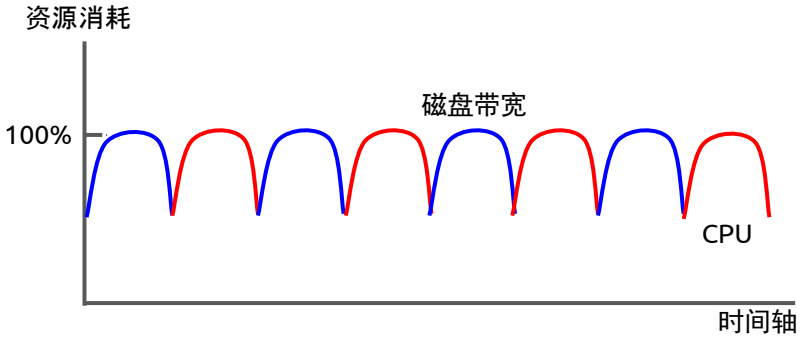
糟糕的应用

Pipeline

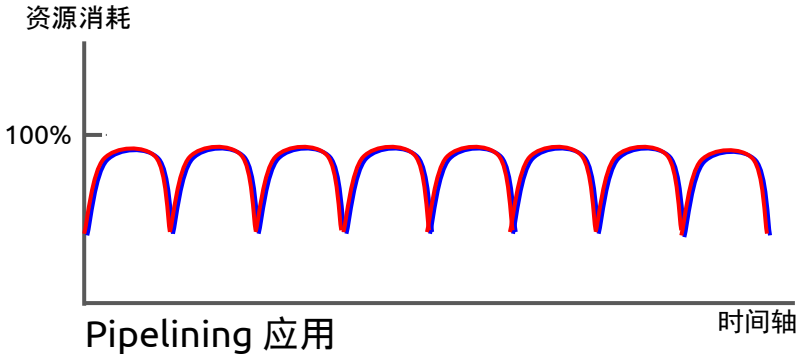
目标：时间轴折叠

- 假设某应用，时间轴和资源消耗为：
  - T0-T1：Encode，消耗 CPU，耗时 100s
  - T1-T2：Write，消耗 IO，耗时 70s
- 划分为10个小事务，每个小事务的时间轴：
  - T0-T1：10s
  - T1-T2：7s
  - 内存占用减少，总体耗时不变，为 170s
- 使用 Pipelining：
  - T0-T1：Encode(Block0)，10s
  - T1-T2：Encode(Block1)，Write(Block0)，10s
  - T2-T3：Encode(Block2)，Write(Block1)，10s
  - 以此类推，总耗时减少至 100s + 7s

Pipeline



CPU 和磁盘消耗交替起伏



CPU、磁盘消耗接近水平线

Pipeline 的 Batch 粒度

- 更小的 batch 粒度
  - “batch 划分”消耗的资源相比更多
    - 锁竞争
    - batch 提交的代价
  - 小事务运行的资源占用更少，整体 资源使用峰值更低
  - 失败代价更低
  - latency 有保障
- 更大的 batch 粒度
  - batch 划分消耗的资源相比更少
  - 整体占用资源多
  - 失败代价大
  - 首、尾 batch 不能从流水线受益
  - 通常可以带来更好的吞吐，但 latency 相对更高
- 平衡的艺术

谨慎决定 batch 粒度

- 例一：于 7200 转的繁忙 HDD 上进行文件拷贝
  - HDD 寻道时间 3-6ms，随机读写最大 QPS 为 400-800
  - HDD 读写带宽 100MB / s
  - 低于 (100MB / 800) ~= 100KB 的 batch 粒度，可能造成性能下降
- 例二：将数据划分小块，并发进行编码
  - 使用 mutex 对数据状态（线程分配等信息）加锁
  - 在高竞争状态，mutex 的 QPS 约为 10-20 万
  - 该编码应用的总体吞吐不高于：10万 * 块大小
  - 若块大小为 4KB，则整体吞吐不高于 400MB / s

常见的瓶颈类型

- CPU
  - CPU usage 高
    - 减少计算开销：更优的算法，减少重复 计算（cache 中间结果），更优的工程实现
  - CPU load 高
    - 线程太多，频繁切换：跨线程交互是否是必要的？
    - 等 IO，top 看到 iowait 比较高：IO 可能是限制性能的瓶 颈
- IO
  - IOPS 上限
    - 减少读写的次数：提高 cache hit ratio
  - IO bandwidth 上限
    - 减少磁盘读写流量：更 紧凑的数据存 储格式，更小的 读写放大(例如只需要 读写 100 字节，结果触发了好多个 page 的读写，从而 产生放大)
  - fsync 次数上限
    - 减少 fsync 次数：合并提交 group commit
- Network
  - bandwidth 上限：降低 传输的数据量（计算下推，更 紧凑的数据格式）
  - send/recv 系统调用太频繁：批量 发送和接收

有用的 Grafana - Node Exporter

PingCAP

PingCAP University

CPU

CPU Profile 原理

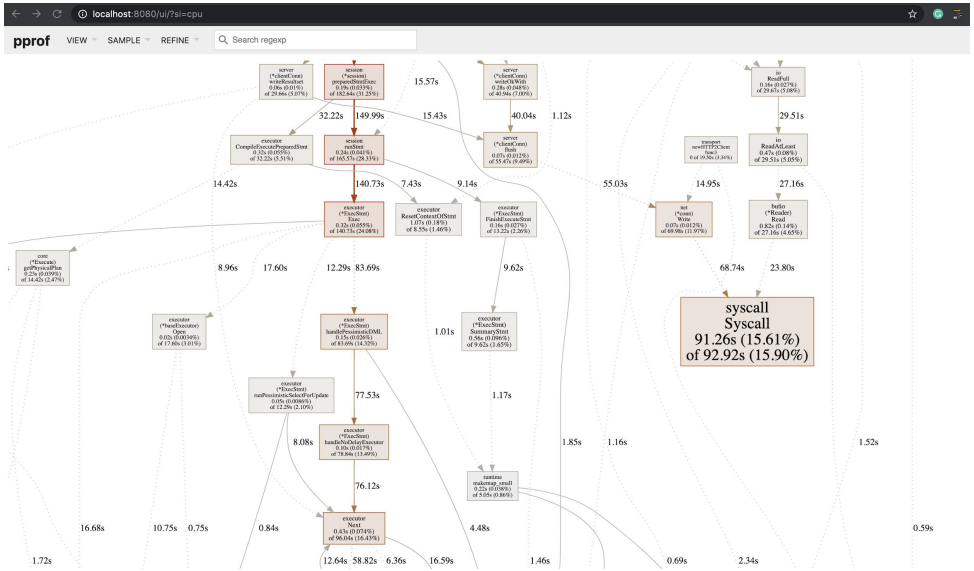
- CPU profile 的原理
  - 在一段时间内对正在执行的程序进行大量的**采样**，每个采样点，是某个线程当前的栈信息，通过对采样的**聚合统计**分析，我们可以得到各个函数的 CPU 资源消耗比例，然后针对性的优化。
  - 采样的时间越长，采样的总数就越多，统计出来的数字就越准确，通常抓取 30 秒，就可以得到一个足够精度的结果。
- 在 Dashboard 的实例性能分析页面，可以方便的对集群内任意实例进行 CPU profile 分析
  - https://docs.pingcap.com/zh/tidb/stable/dashboard-profiling

Profile TiDB CPU usage(1/3)

- curl http://{TiDBIP}:10080/debug/zip?seconds=60 --output debug.zip，会把分析 tidb 性能问题所需的多个 profile 文件打包生成了一个 zip 文件
- debug.zip 解压后可以看到以下几个文件：
  - - version
  - tidb-server 版本
  - - profile
  - CPU profile 文件
  - - heap
  - heap profile 文件
  - - goroutine
  - 所有 goroutine 的栈信息
  - - mutex
  - mutex profile 文件
  - - config
  - tidb server 的配置文件

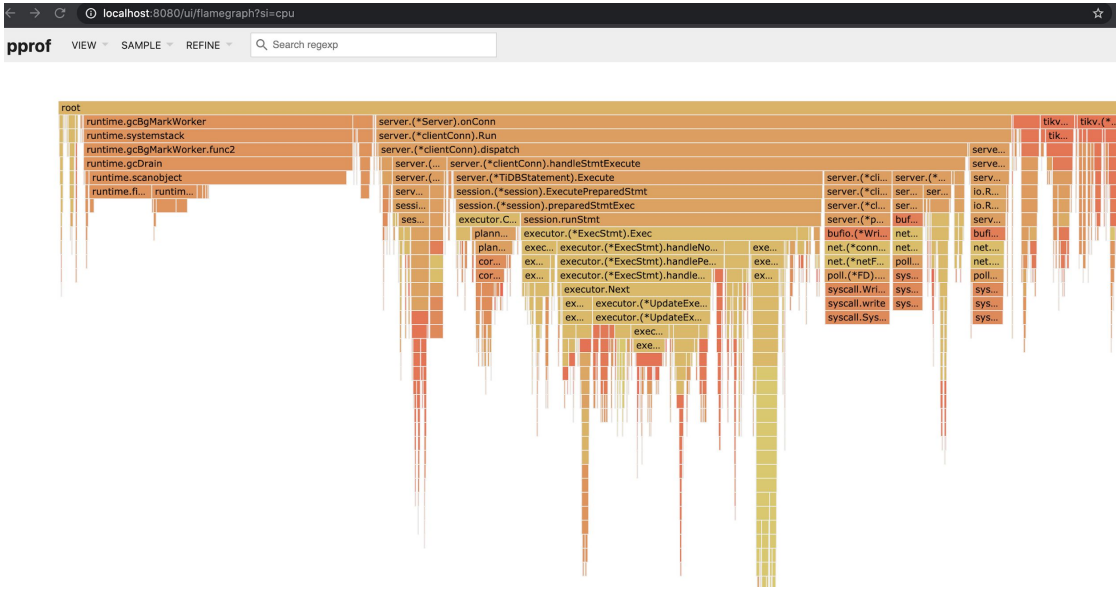
Profile TiDB CPU usage(2/3)

- 我们暂时只关心 profile 文件，通过下面命令在 web 界面上查看：go tool pprof -http=:8080 debug/profile
- 在 Graph 里，每一个**函数节点**都是**唯一**的，有多个指向它的调用
- Graph 格式让我们比较容易发现开销大的子函数，指引我们对子函数本身做优化。
- 在 Graph 格式里，函数节点除了函数名称，还包含了两个时间，前面的时间是函数自身消耗的 CPU 时间，后面的时间是函数本身 + 调用的子函数消耗的整体的 CPU 时间
- 函数本身 CPU 时间越长，函数节点面积就越大，函数整体 CPU 时间越长，节点的颜色就越深。
- 选择一个函数节点，节点颜色会变成紫色。这时选择 Refine -> Focus，可以让整个 graph 只展示和这个节点相关的 CPU 开销。



Profile TiDB CPU usage(3/3)

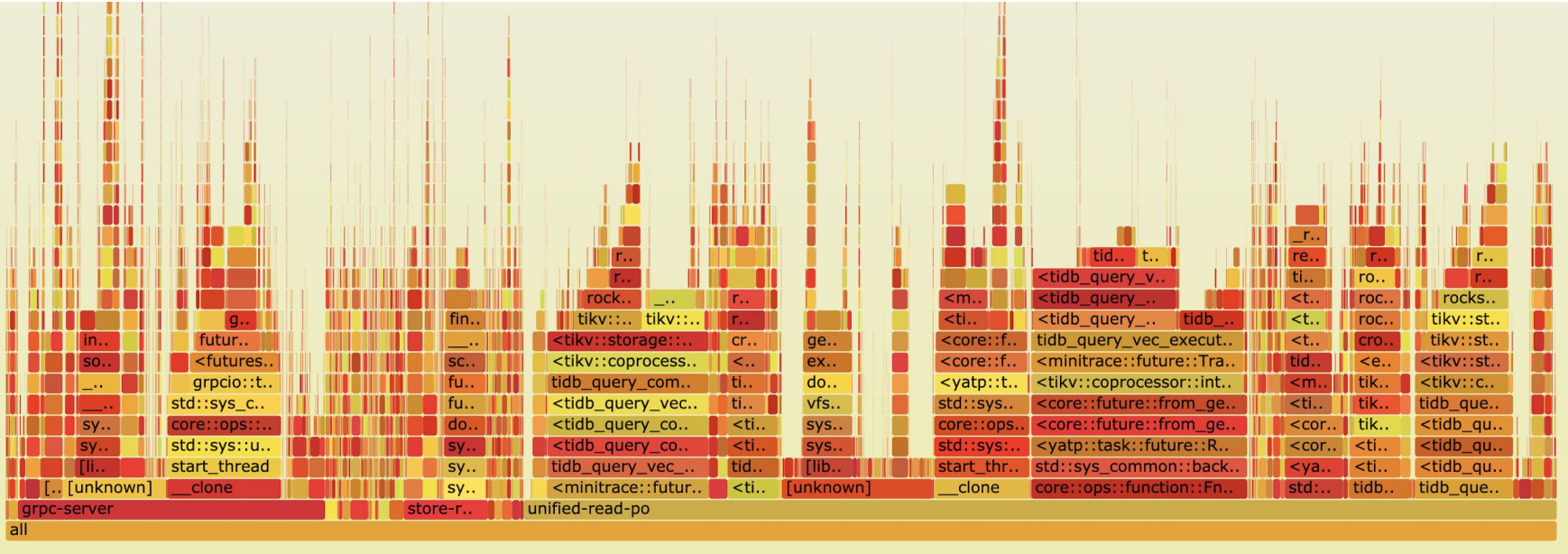
- 通过点击左上角的 VIEW 可以以 FlameGraph 的方式展示
- 在 FlameGraph 里，子函数的开销是分散在不同的父函数里的。
- FlameGraph 格式会让我们从整体上看到各个线程的开销比例和子函数占用的比例，指引我们从整体上找到优化的优先级。
- Graph 和 FlameGraph 都可以方便的让我们找到函数调用关系，指导我们阅读代码



Profile TiKV CPU usage(1/3)

- 方式1：TiDB Dashboard -> 高级调试 -> 实例性能分析 -> 浏览器查看 svg 文件
- 方式2：perf 命令
  - git clone https://github.com/pingcap/tidb-inspect-tools
  - cd tracing_tools/perf
  - sudo sh ./**cpu_tikv.sh** $tikv-pid
  - sudo sh ./**dot_tikv.sh** $tikv-pid

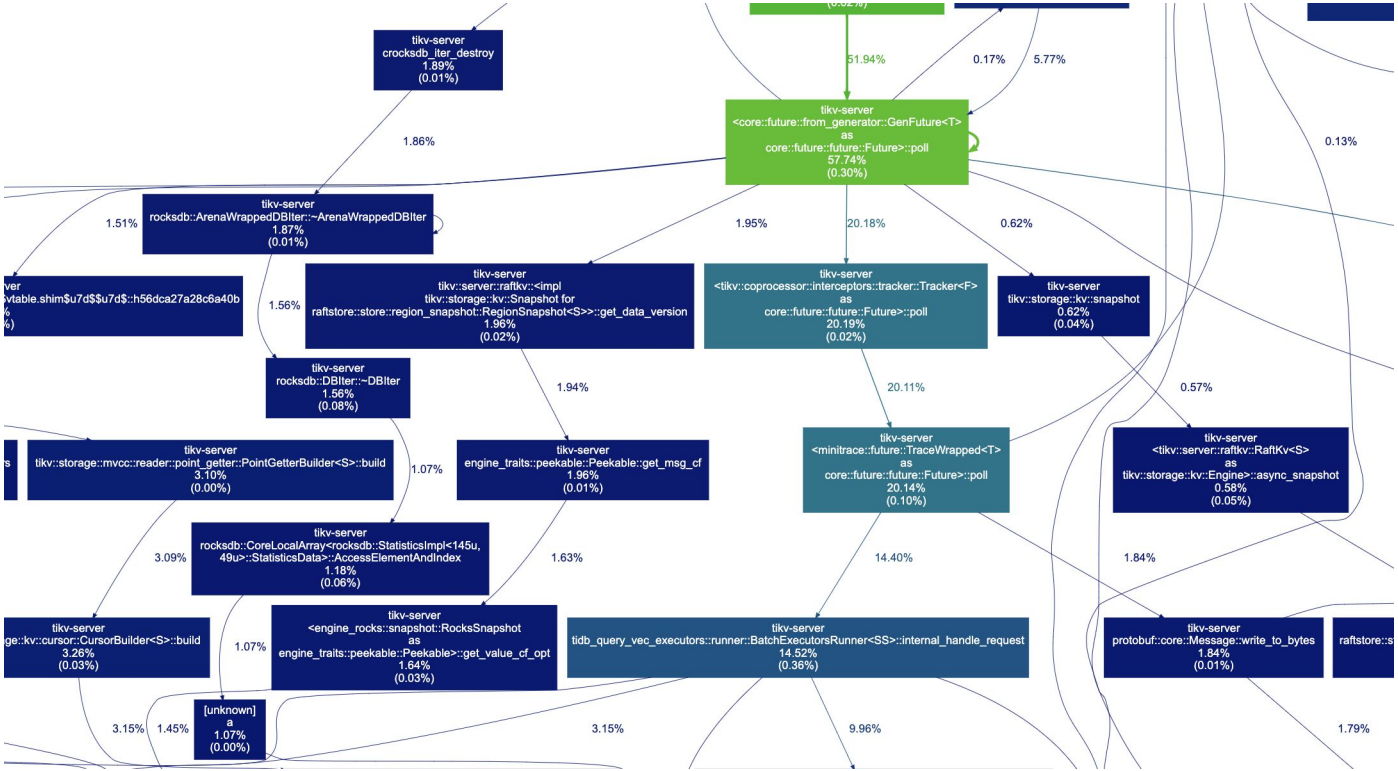
Profile TiKV CPU usage(2/3)



PingCAP

PingCAP University

Profile TiKV CPU usage(3/3)

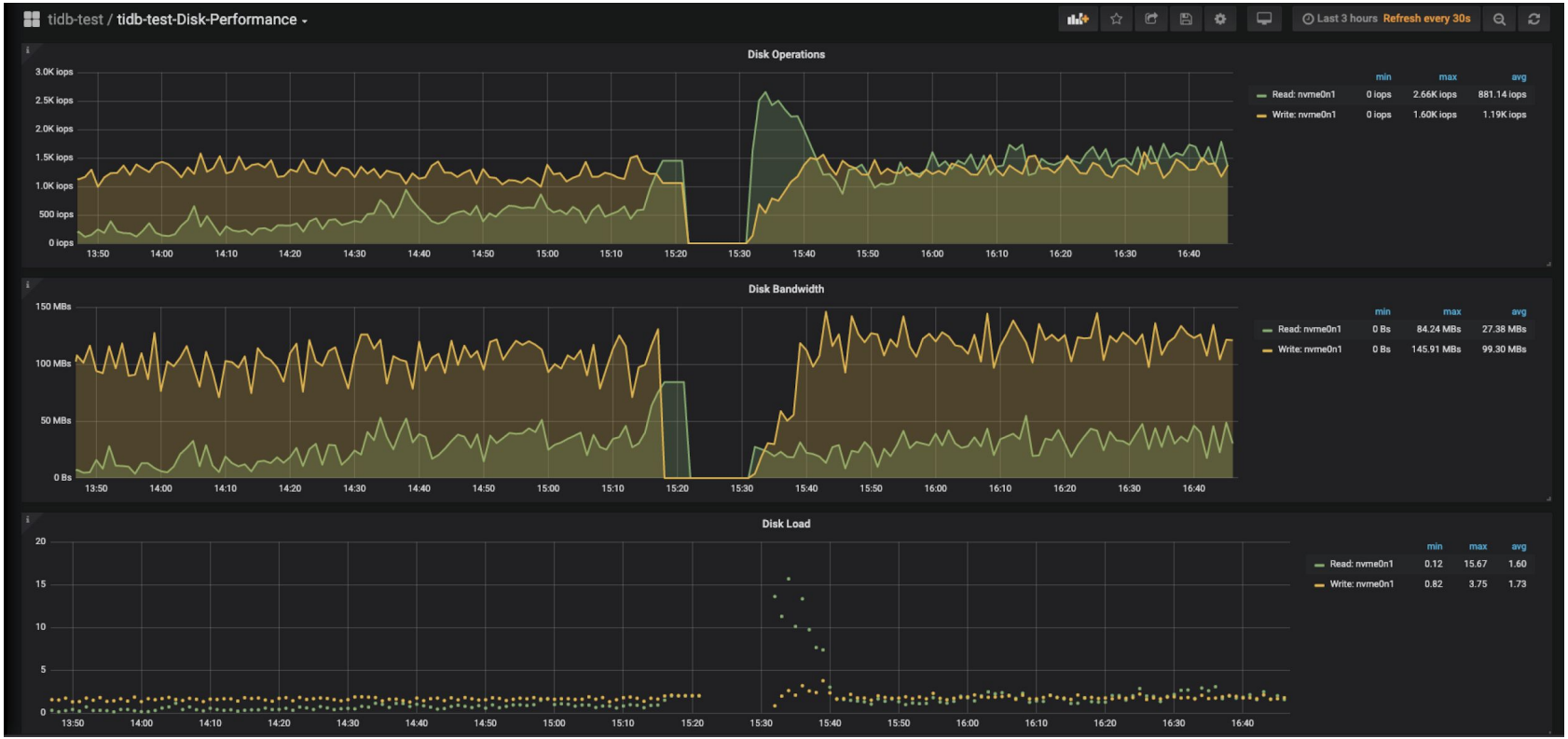


PingCAP

PingCAP University

IO

Grafana: Disk-Performance



iostat

- iostat -x 1 -m
  - Device: rrqm/s wrqm/s r/s w/s rMB/s wMB/s avgrq-sz avgqu-sz await r_await w_await svctm %util
  - nvme0n1 0.00 0.00 0.00 5.00 0.00 0.02 8.00 0.00 0.00 0.00 0.00 0.00 0.00
  - sda 0.00 15.00 0.00 30.00 0.00 0.20 13.60 0.00 0.07 0.00 0.07 0.07 0.20
  - sdb 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
- await：平均每次设备 I/O 操作的等待时间 (毫秒)
- r_await：读操作等待时间
  - 如果该值比较高（比如超过 5ms 级别）说明磁盘读压力比较大
- w_await：写操作等待时间
  - 如果该值比较高（比如超过 5ms 级别）说明磁盘写压力比较大
- 可以看 io size、io 连续性、读写瞬时流量、读写分别的 iops

iotop

- iostat 是从盘的角度出发来看 IO
- iotop 是从进程角度看 IO
  - iotop 用于看各个线程的 io 累积量，有没有超出预期，顺便作为 fsync 不足的佐证(jbd2流量超大)
  - sudo iotop -o
    - o:只显示有IO输出的进程。

```
Total DISK READ :       0.00 B/s | Total DISK WRITE :       6.97 K/s
Actual DISK READ:       0.00 B/s | Actual DISK WRITE:     167.38 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
  986 be/3 root        0.00 B/s    0.00 B/s  0.00 %  0.01 % [jbd2/nvme0n1-8]
48615 be/4 pingcap     0.00 B/s    0.00 B/s  0.00 %  0.00 % bin/blackbox_exporter/blackbox_exporter --web.listen-address=:9335
49008 be/4 pingcap     0.00 B/s    0.00 B/s  0.00 %  0.00 % bin/blackbox_exporter/blackbox_exporter --web.listen-address=:9335
57630 be/4 pingcap     0.00 B/s    0.00 B/s  0.00 %  0.00 % bin/blackbox_exporter/blackbox_exporter --web.listen-address=:9335
167673 be/4 tidb       0.00 B/s    0.00 B/s  0.00 %  0.00 % bin/blackbox_exporter/blackbox_exporter --web.listen-address=:9120
48899 be/4 pingcap     0.00 B/s    0.00 B/s  0.00 %  0.00 % bin/blackbox_exporter/blackbox_exporter --web.listen-address=:9335
49249 be/4 pingcap     0.00 B/s    0.00 B/s  0.00 %  0.00 % bin/blackbox_exporter/blackbox_exporter --web.listen-address=:9335
 6228 be/4 zhangxin    0.00 B/s    3.49 K/s  0.00 %  0.00 % tidb-server -P 4000 --store=tikv --host=127.0.0.1 --status=10080 --
48502 be/4 pingcap     0.00 B/s    3.49 K/s  0.00 %  0.00 % bin/prometheus/prometheus --config.file=/home/pingcap/cs/cluster/de
```

iosnoop

- iosnoop
  - http://www.brendangregg.com/blog/2014-07-16/iosnoop-for-linux.html
  - 磁盘延迟毛刺
    - iosnoop -ts [-d device] [-p PID] [-i iotype]

```
# ./iosnoop -ts > out
# more out
Tracing block I/O. Ctrl-C to end.
STARTs          ENDs            COMM         PID    TYPE DEV      BLOCK      BYTES   LATms
6037559.121523 6037559.121685 randread     22341  R    202,32   29295416   8192     0.16
6037559.121719 6037559.121874 randread     22341  R    202,16   27515304   8192     0.16
[...]
6037595.999508 6037596.000051 supervise    1692   W    202,1    12862968   4096     0.54
6037595.999513 6037596.000144 supervise    1687   W    202,1    17040160   4096     0.63
6037595.999634 6037596.000309 supervise    1693   W    202,1    17040168   4096     0.68
6037595.999937 6037596.000440 supervise    1693   W    202,1    17040176   4096     0.50
6037596.000579 6037596.001192 supervise    1689   W    202,1    17040184   4096     0.61
6037596.000826 6037596.001360 supervise    1689   W    202,1    17040192   4096     0.53
6037595.998302 6037596.018133 randread     22341  R    202,32   954168     8192    20.03
6037595.998303 6037596.018150 randread     22341  R    202,32   954200     8192    20.05
6037596.018182 6037596.018347 randread     22341  R    202,32   18836600   8192     0.16
[...]
```

其他 IO 工具

- fio 可以用于测我们比较关注的三个磁盘的重要指标：读写带宽、IOPS、fsync / 每秒，另外还可以获得 io 延迟分布
  - fsync = n，n 次 write 之后调用一次 fsync
- pg_test_fsync 用于测 fsync 性能，简单易用

Memory

Profile TiDB Memory

- 查看 in-use 内存
  - go tool pprof -http=:8080 debug/heap
  - 从 heap profile 图里，我们可以看 in-use 的内存，都是由哪些函数分配出来的
- 查看历史上 alloc 的 space
  - go tool pprof -alloc_space -http=:8080 debug/heap
  - 在 profile heap 的时候指定 -alloc_space，可以看到总共分配过的内存，即使已经被释放了
- 查看 mutex 竞争情况
  - go tool pprof -contentions -http=:8080 debug/mutex

Profile TiKV Memory

- 方式1：perf 命令
  - https://github.com/pingcap/tidb-inspect-tools/tree/master/tracing_tools/perf
  - cd tracing_tools/perf
  - sudo sh **mem.sh** $tikv-pid
  - 针对 TiKV 需要做一些事情 *sudo perf probe -x {tikv-binary} -a malloc*，然后用 *perf record -e probe_tikv:malloc -F 99 -p $1 -g -- sleep 10* 替代 mem.sh 的第一个命令
- 方式2：bcc 工具
  - Linux 4.9 及以上版本
  - sudo /usr/share/bcc/tools/stackcount -p $tikv-pid -U $tikv-binary-path:malloc > out.stacks
  - ./stackcollapse.pl < out.stacks | ./flamegraph.pl --color=mem \
  - --title="malloc() Flame Graph" --countname="calls" > out.svg
  - http://www.brendangregg.com/FlameGraphs/memoryflamegraphs.html
- 方式3：jemalloc 统计信息
  - tiup ctl tikv --host=$tikv-ip:$tikv-status-port metrics --tag=jemalloc
  - tiup ctl:nightly tikv --host=$tikv-ip:$tikv-status-port metrics --tag=jemalloc

一个好用的 Profile 工具 - VTune

What is VTune?

Intel® VTune™ Profiler is a **performance analysis tool** for users who develop serial and multithreaded applications. VTune Profiler helps you analyze the algorithm choices and identify where and how your application can benefit from available **hardware** resources.

Use VTune Profiler to locate or determine:

- The most time-consuming (**hot**) functions in your application and/or on the whole system
- Sections of code that do not effectively utilize available processor time
- The best sections of code to optimize for sequential performance and for threaded performance
- **Synchronization objects** that affect the application performance
- Whether, where, and why your application spends time on input/output operations
- The performance impact of different **synchronization methods**, different numbers of threads, or different algorithms
- Thread activity and transitions
- Hardware-related issues in your code such as **data sharing**, **cache misses**, **branch misprediction**, and others

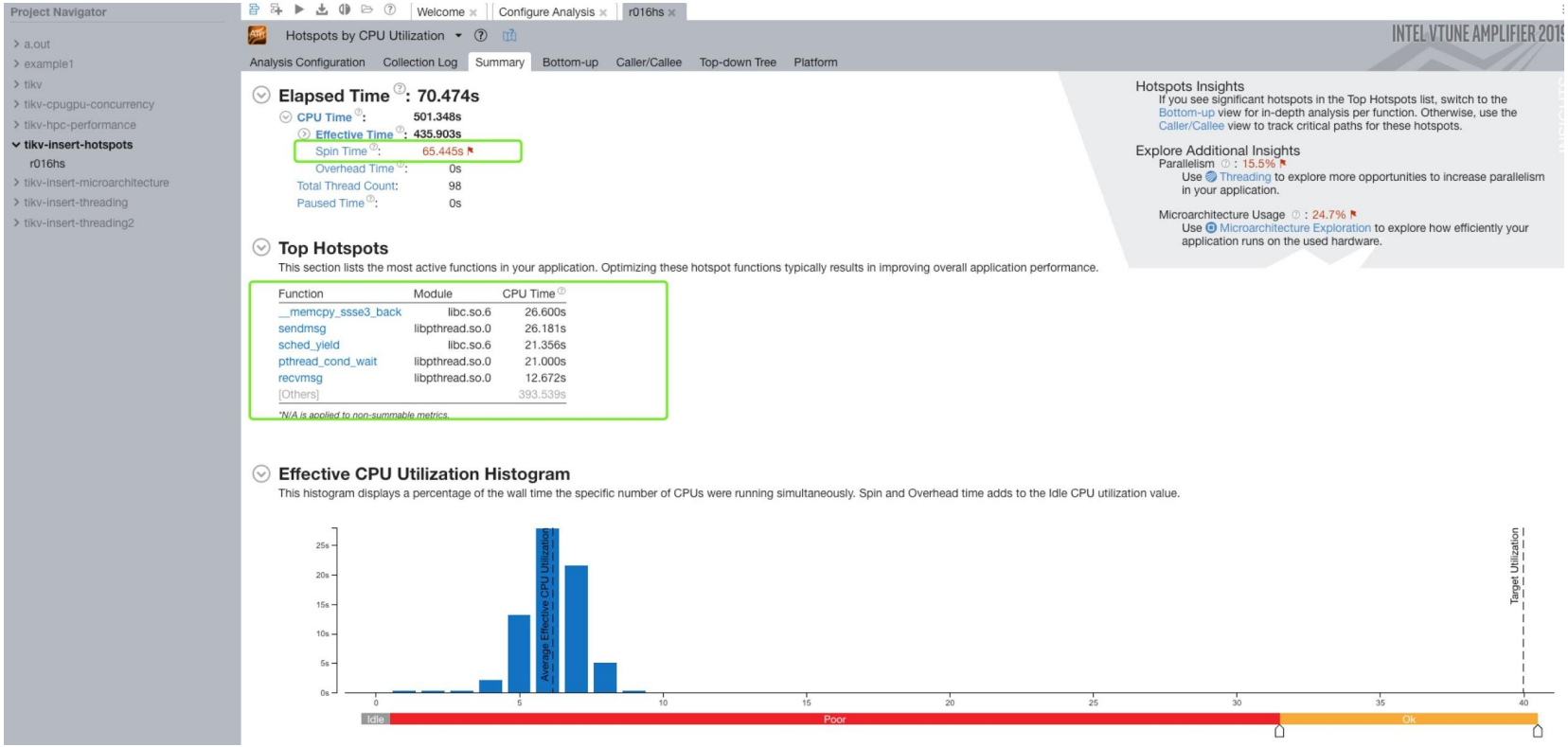
VTune 核心功能（1/2）

- **Hotspots Analysis** for CPU usage isues
- **Threading Analysis**: Use the Threading analysis to identify how efficiently an application uses available processor compute cores and explore inefficiencies in threading runtime usage or contention on threading synchronization that makes threads waiting and prevents effective processor utilization.
- **Microarchitecture Exploration Analysis** for Hardware Issues. You can perform Microarchitecture Exploration analysis to understand how efficiently your code is passing through the core pipeline.
- **Memory Access Analysis** for Cache Misses and High Bandwidth Issues. to identify memory-related issues, like NUMA problems and bandwidth-limited accesses, and attribute performance events to memory objects (data structures), which is provided due to instrumentation of memory allocations/de-allocations and getting static/global variables from symbol information.

VTune 核心功能(2/2)

- **Memory Consumption Analysis**: Use the Memory Consumption analysis for your Linux* native or Python* targets to explore memory consumption (RAM) over time and identify memory objects allocated and released during the analysis run.
- **System Overview Analysis**: Use a platform-wide System Overview analysis to monitor a general behavior of your target Linux* or Android* system and correlate power and performance metrics with the interrupt request (IRQ) handling.
- **IO Analysis**: Use a platform-wide Input and Output analysis to monitor utilization of the disk and network subsystems, CPU and processor buses.

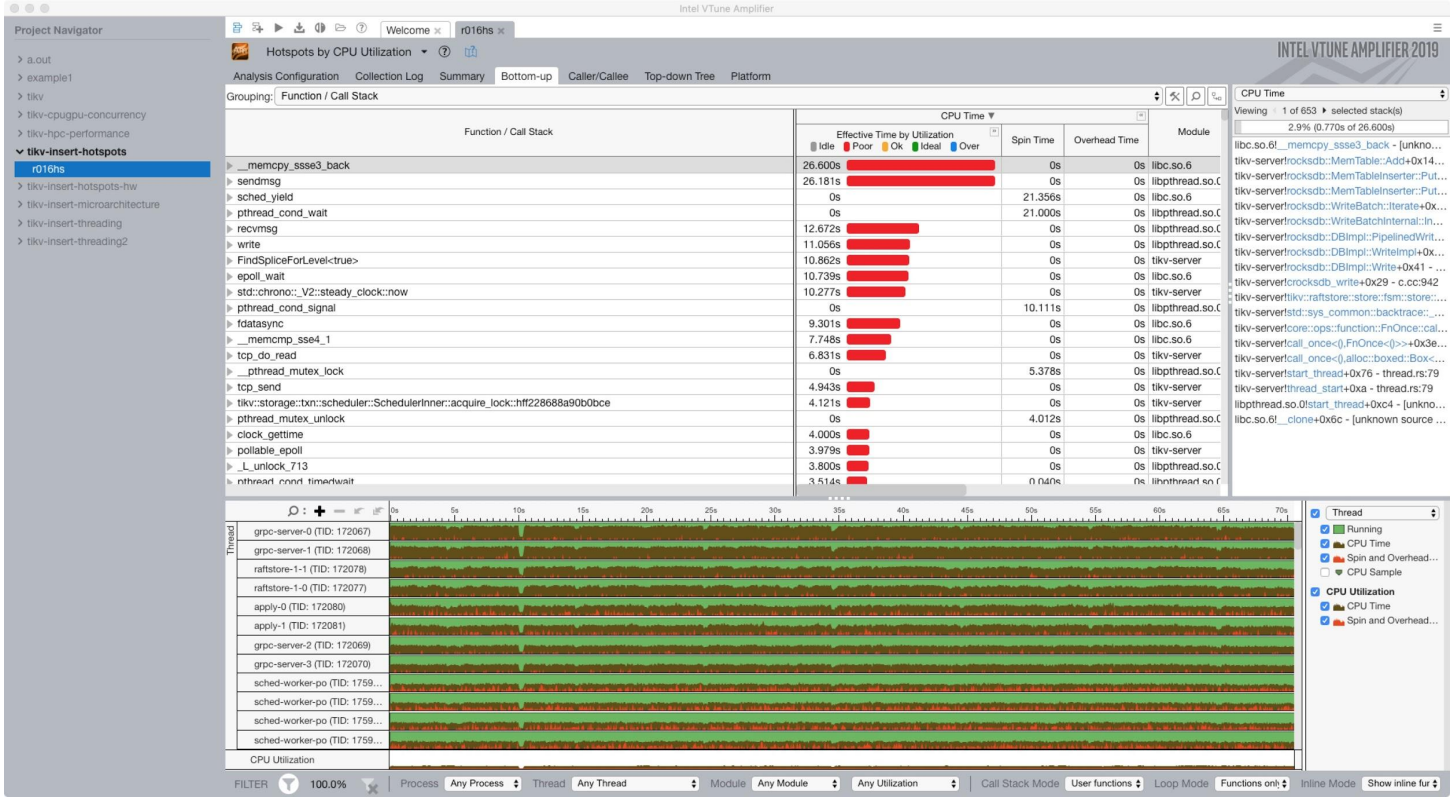
Hotspots - summary



PingCAP

PingCAP University

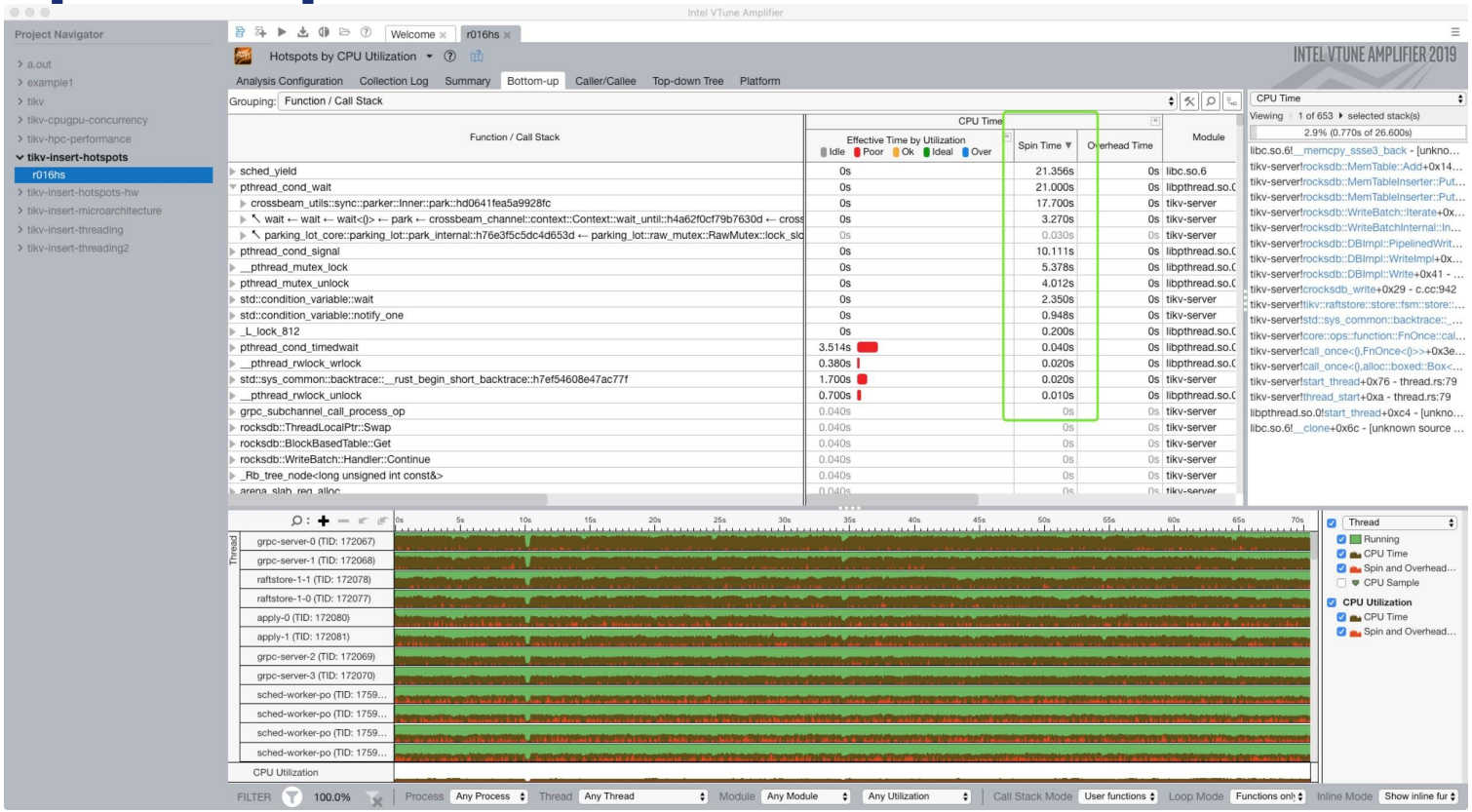
Hotspots - CPU time



PingCAP

PingCAP University

Hotspots - spin time



PingCAP

PingCAP University

Try it yourself

作业

课程作业

分值：1 个有效 issue 100，有效 PR 根据实际效果进行相应加分，比如能节省 CPU、减少内存占用、减少 IO 次数等。

题目描述：

使用上一节可以讲的 sysbench、go-ycsb 或者 go-tpc 对 TiDB 进行压力测试，然后对 TiDB 或 TiKV 的 CPU 、内存或 IO 进行 profile，寻找潜在可以优化的地方并提 enhance 类型的 issue 描述。

issue 描述应包含：

- 部署环境的机器配置（CPU、内存、磁盘规格型号），拓扑结构（TiDB、TiKV 各部署于哪些节点）
- 跑的 workload 情况
- 对应的 profile 看到的情况
- 建议如何优化？

【可选】提 PR 进行优化：

- 按照 PR 模板提交优化 PR

输出：对 TiDB 或 TiKV 进行 profile，写文章描述分析的过程，对于可以优化的地方提 issue 描述，并将 issue 贴到文章中（或【可选】提 PR 进行优化，将 PR 贴到文章中）

截止时间：9.1 24:00（逾期提交不给分）

作业提交方式

- 作业提交方式：提交至个人 GitHub，发送邮件至 talent-plan@tidb.io
- 邮件主题建议：HP - Lesson N-【GitHub ID】
  - N 代表课程章节周数，例如第三 节课程的作业邮件主题为 HP- Lesson 03-【GitHub ID】）
- 邮件正文建议：
  - 作业提交文档内容包含两部分：
    - Github 链接
    - 性能分析的过程，潜在的瓶颈点，潜在的优化手段
- 友情提示：请不要直接在邮件中添加压缩包（GitHub 链接为唯一格式）

课程答疑与学习反馈

扫描左侧二维码填写报名信息，加入课程学习交流群，课程讲师在线答疑，学习效果 up up！

更多课程

想要了解更多关于 TiDB 运维、部署以及 TiDB 内核原理相关课程，可以扫描左侧二维码，或直接进入 university.pingcap.com 查看

Thank you!