阿里云 | 奥运会全球指定云服务商

io_uring 优化及应用实践

王小光/齐江

阿里云基础软件部-操作系统

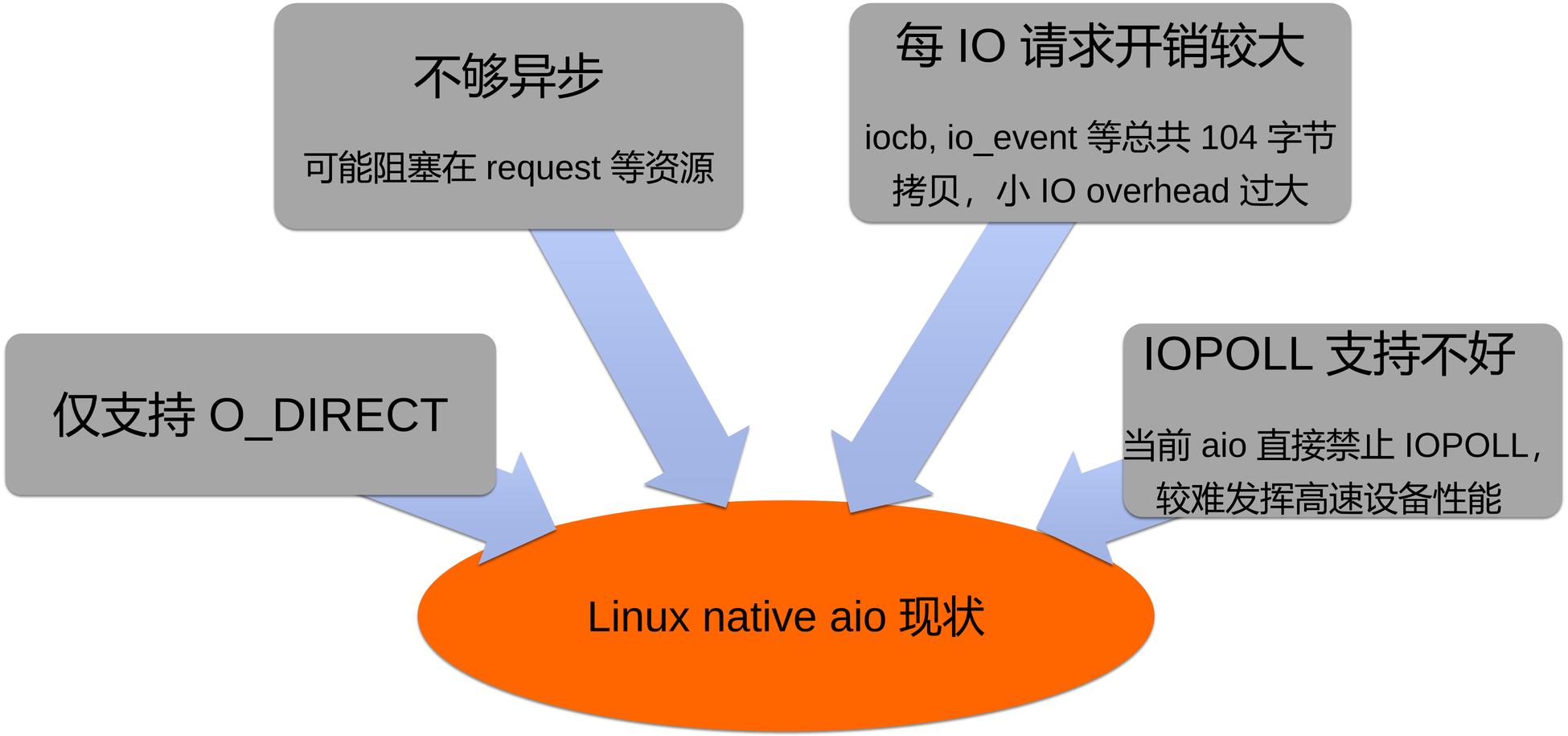
2020/10/14

Agenda

阿里云 | 奥运会全球指定云服务商

- 为什么需要 io_uring
- io_uring 架构及重要特性
- io_uring 优化工作
- io_uring 应用实践
- 下一步工作

为什么需要 io_uring



Agenda

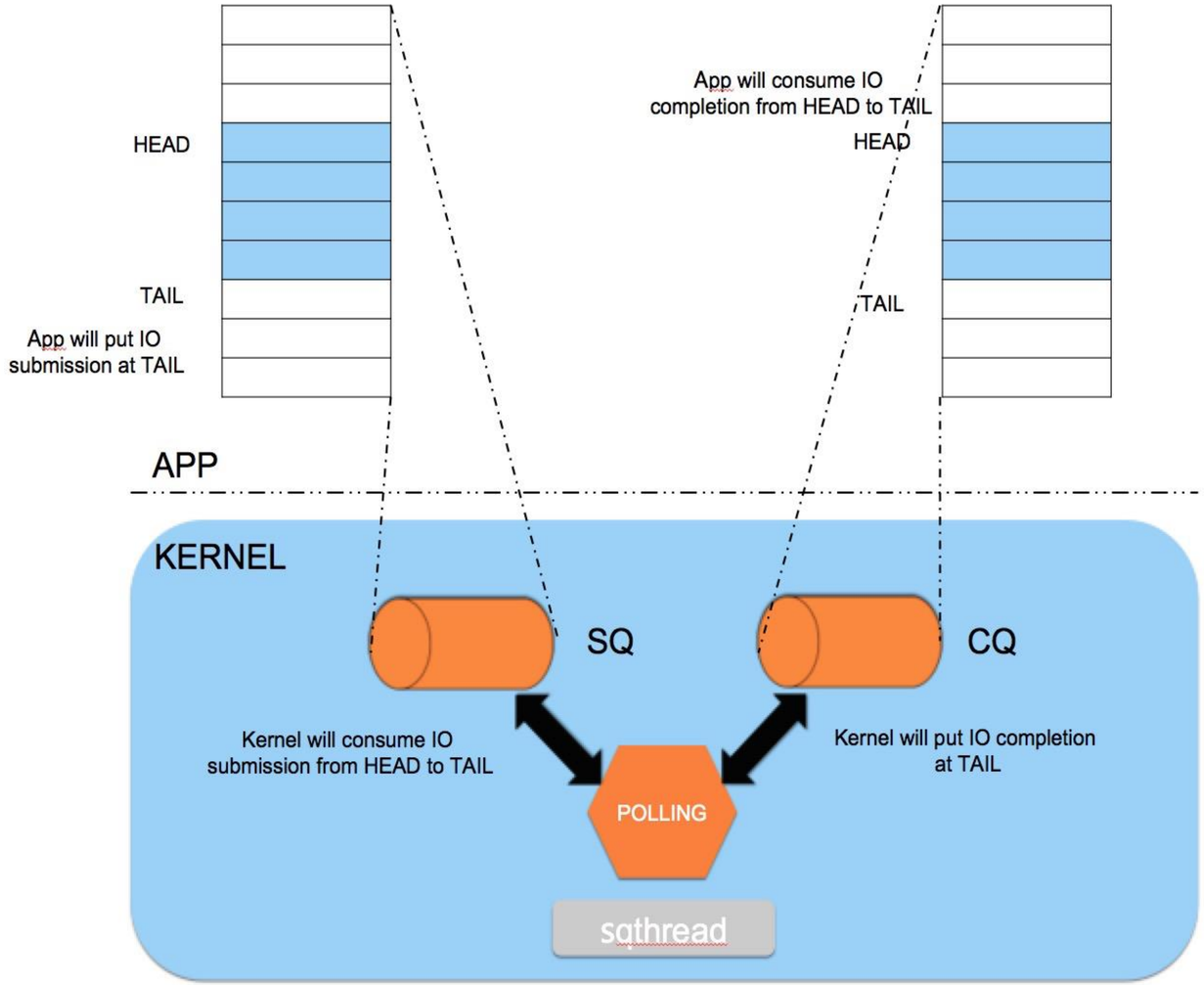
阿里云 | 奥运会全球指定云服务商

- 为什么需要 io_uring
- io_uring 架构及重要特性
- io_uring 优化工作
- io_uring 应用实践
- 下一步工作

io_uring 设计目标

- 使用方便
  - 简单且强大的系统调用，共提供三个系统调用，liburing 用户态库编程友好
- 通用性强
  - 提供内核统一的异步编程框架，既支持传统 FS / Block IO, 也支持网络编程
- 特性丰富
  - 支持非常多的高级特性，有丰富的想象空间
- 高性能
  - IO 请求 overhead 尽可能小，消除目前 libaio 存在每请求104 字节的拷贝

io_uring 整体架构



- 两组 ring buffer，一组用于存放 sqe，用于描述 IO 请求提交；一组用于存放 cqe，用于描述 IO 请求完成。
- io_uring_enter(2) 既可以提交 IO 请求，也可以 reap 完成的 IO 请求。

io_uring 重要特性

- IORING_SETUP_SQPOLL

  - 创建一个内核线程进行 sqe 的处理（IO 提交），几乎完全消除用户态内核态上下文切换，消除spectre / meltdown 缓解场景下对系统调用的性能影响，此特性需要额外消耗一个 cpu 核。

- IORING_SETUP_IOPOLL

  - 配合 blk-mq 多队列映射机制，内核 IO 协议栈开始真正完整支持 IO polling。

- IORING_REGISTER_FILES / IORING_REGISTER_FILES_UPDATE / IORING_UNREGISTER_FILES

  - 减少 fget / fput 原子操作带来的开销。

io_uring 重要特性

- IORING_REGISTER_BUFFERS / IORING_UNREGISTER_BUFFERS

  - 通过提前向内核注册 buffer，减少 get_user_pages/ unpin_user_pages 开销，应用适配存在一定难度。

- IORING_FEAT_FAST_POLL

  - 网络编程新利器，向 epoll 等传统基于事件驱动的网络编程模型发起挑战，为用户态提供真正的异步编程 API。

- ASYNC BUFFERED READS

  - 更好的支持异步 buffered reads，但当前对于异步 buffered write 的支持还不够完美。

Agenda

阿里云 | 奥运会全球指定云服务商

- 为什么需要 io_uring
- io_uring 架构及重要特性
- io_uring 优化工作
- io_uring 应用实践
- 下一步工作

io_uring 优化工作

Alibaba Cloud Linux 2（4.19 内核）已支持该特性，我们在实践过程中针对性进行了优化，相关优化补丁都已贡献到上游社区。

截止目前累计贡献近 40 个补丁，涵盖特性支持和重构，性能优化，bugfix 等。

- 特性支持和重构
  - 聚焦 sqpoll 和 iopoll 场景，打通 ext4/block/nvme driver 的 polling 流程
  - 重构 file register/unregister/update 特性，优化大量文件场景的使用
  - …

io_uring 优化工作

- 性能优化
  - 优化 io_uring 框架自身的开销
  - 减少 50% liburing 系统调用数量
  - IO 提交性能优化 30%
  - …
- bugfix
  - iopoll 场景下多个 deadlock，race
  - cqe ring buffer 与 liburing 状态同步
  - io_uring fast poll 场景下 panic
  - …

io_uring 性能测试框架

- io_uring 作为一个比较新的技术，社区火热，开发节奏非常快
- 如何在高节奏开发中快速识别 regression?
  - liburing 自带的测试集主要覆盖功能
- 开发 perf-test-for-io_uring 测试套件
  - 及时发现性能回退问题
  - 已在 openanolis.org 社区开源

io_uring 优化工作——进行中

- io_uring_enter(2) timeout 功能支持

  - io_uring_enter(2) 可以用来接收已经完成的 IO 请求，但没有提供类似 io_getevents(2) 中的超时功能。

  - 目前如果用户需要以超时方式等待 IO 完成，需要先发送一个 IORING_OP_TIMEOUT 类型的 sqe，此方式需要在 io_uring 的 sqe ring buffer 上进行同步，用户态使用不方便，且非常影响性能。

  - 在 io_uring_enter(2) 中添加 timeout 支持，在 nop opcode 测试下有 2X 的性能提升。

io_uring 优化工作——进行中

- 多个 io_uring 实例共享同一个 sqthread 内核线程

  - 开启 sqpoll 特性的 io_uring 实例会创建专属内核线程，该内核线程会一直 polling IO。

  - 将不同实例的内核线程绑定到不同的核，严重浪费 cpu 资源。

  - 绑定到同样的核，内核线程间无序的竞争会严重　影响 IO 性能。

  - 采用 Round Robin 形式，多个 io_uring 实例共享一个内核线程，既节省了资源，又不影响性能。

No of instances	1	2	4	8	16	32
Unpatched (IOPS)	589k	487k	303k	165k	85.8k	43.7k
Patched (IOPS)	590k	593k	581k	538k	494k	406k

No of instances	1	2	4	8	16	32
Unpatched (Lat)	217	262	422	775	1488	2917
Patched (Lat)	216	215	219	237	258	313

Agenda

阿里云 | 奥运会全球指定云服务商

- 为什么需要 io_uring
- io_uring 架构及重要特性
- io_uring 优化工作
- **io_uring 应用实践**
- 下一步工作

echo server

echo server 评估模型中，服务端接收到消息即原样返回给客户端。

传统上用 epoll 来实现 echo server，我们利用IORING_FEAT_FAST_POLL 特性将其重写。

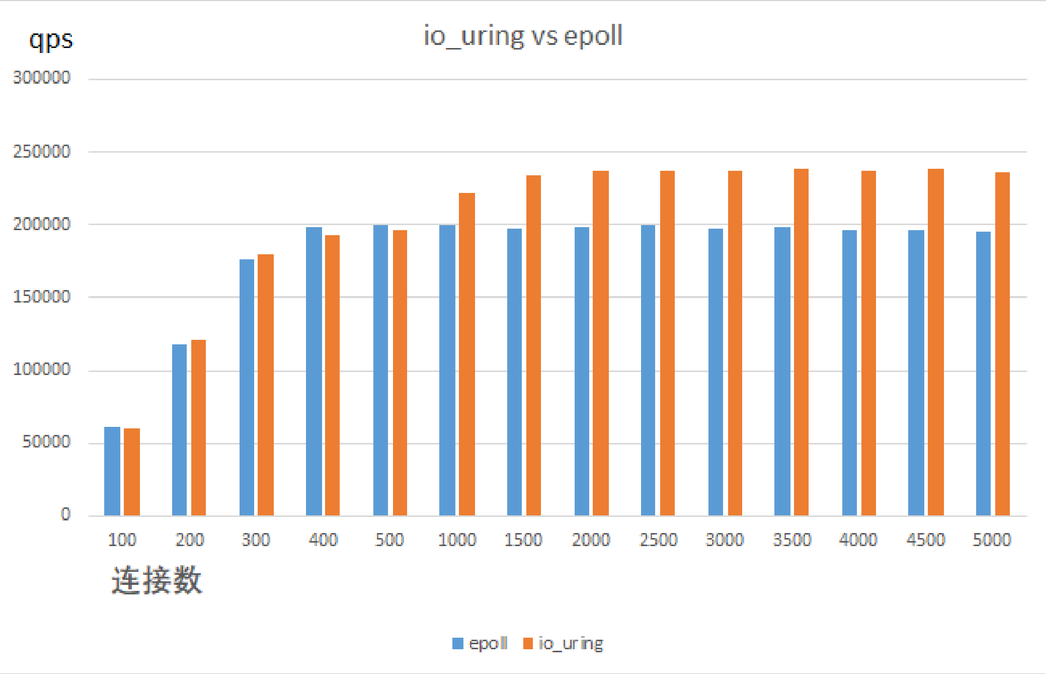
```
struct epoll_event ev;

/* for accept(2) */
ev.events = EPOLLIN;
ev.data.fd = sock_listen_fd;
epoll_ctl(epollfd, EPOLL_CTL_ADD, sock_listen_fd, &ev);

/* for recv(2) */
ev.events = EPOLLIN | EPOLLET;
ev.data.fd = sock_conn_fd;
epoll_ctl(epollfd, EPOLL_CTL_ADD, sock_conn_fd, &ev);

new_events = epoll_wait(epollfd, events, MAX_EVENTS, -1);
for (i = 0; i < new_events; ++i) {
    /* process every events */
    ...
}
```

```
/* 用sqe对一次recv操作进行描述 */
struct io_uring_sqe *sqe = io_uring_get_sqe(ring);
io_uring_prep_recv(sqe, fd, bufs[fd], size, 0);

/* 提交该sqe，也就是提交recv操作 */
io_uring_submit(&ring);

/* 等待完成的事件 */
io_uring_submit_and_wait(&ring, 1);
cqe_count = io_uring_peek_batch_cqe(&ring, cqes, sizeof(cqes) / sizeof(cqes[0]));
for (i = 0; i < cqe_count; ++i) {
    struct io_uring_cqe *cqe = cqes[i];
    /* 依次处理reap每一个io请求，然后可以调用请求对应的handler */
    ...
}
```

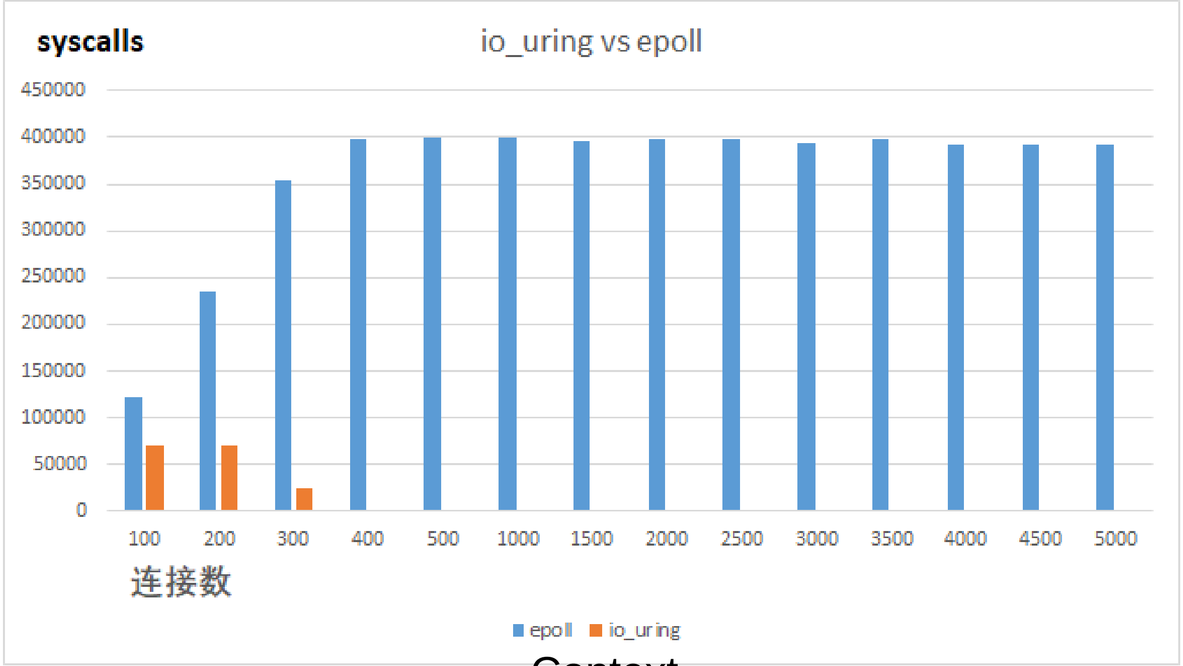
echo server 优化效果

阿里云 | 奥运会全球指定云服务商

- 当连接数 1000 及以上时，io_uring 的性能优势开始体现。io_uring 的极限性能单 core 在 240k qps 左右，而 epoll 单 core 只能达到 200k qps 左右，整体优化约 20%。
- io_uring 可以极大的减少用户态到内核态的切换次数，在连接数超过 300 时 io_uring 用户态到内核态的切换次数基本可以忽略不计。



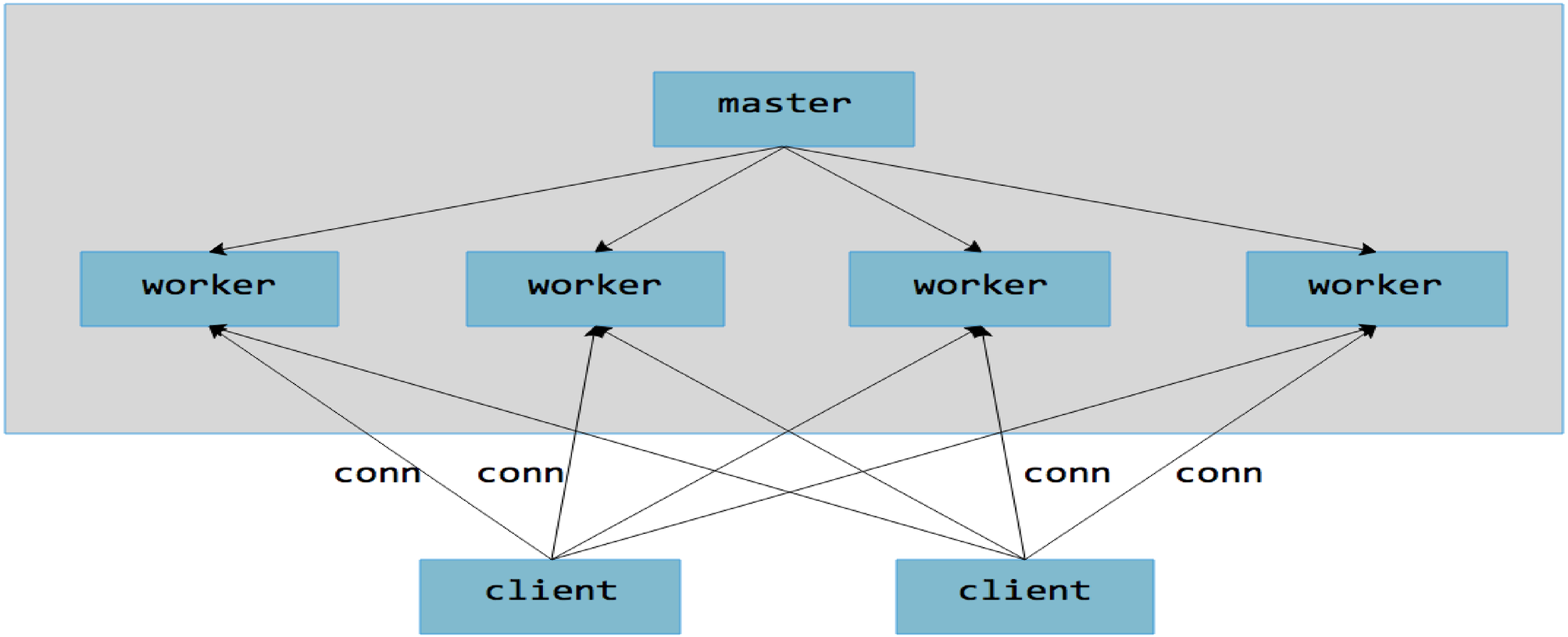
QPS



Context Switch

Nginx

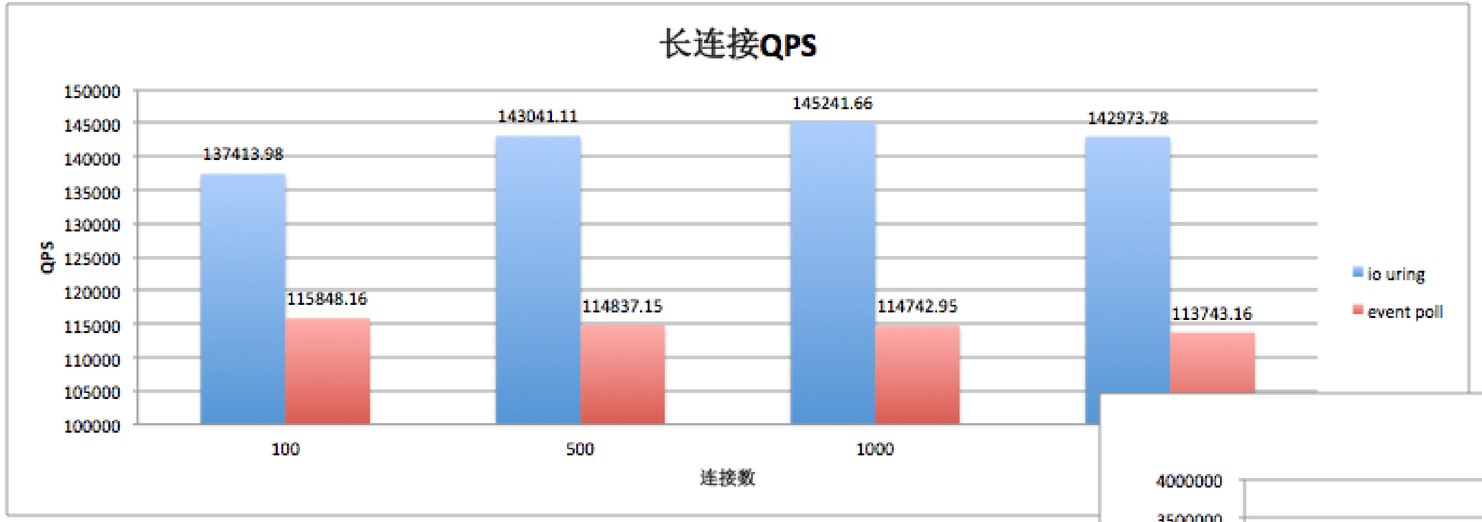
Nginx 是基于 epoll，我们将其用 io_uring 进行改写，利用 io_uirng 的 IORING_FEAT_FAST_POLL 特性。

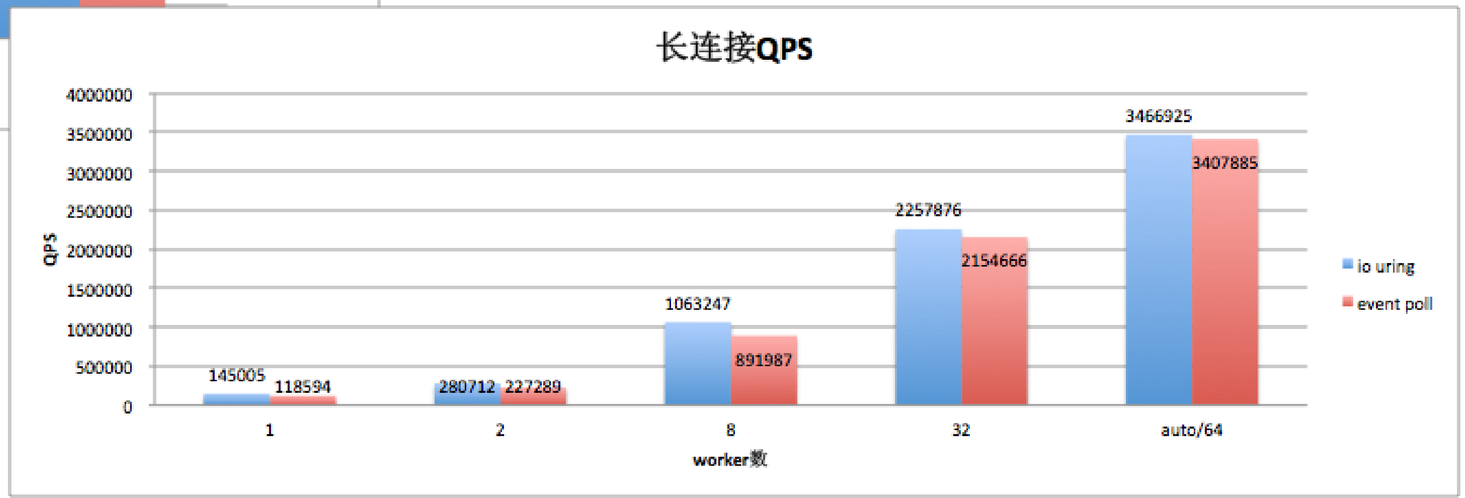


Nginx 优化效果

阿里云 | 奥运会全球指定云服务商

- 单 worker 场景，当连接数超过 500 时，QPS提升 20% 以上。
- 连接数固定 1000，worker 数目在 8 以下时，QPS 有 20% 左右的提升。随着 worker 数目增大，收益逐渐降低。

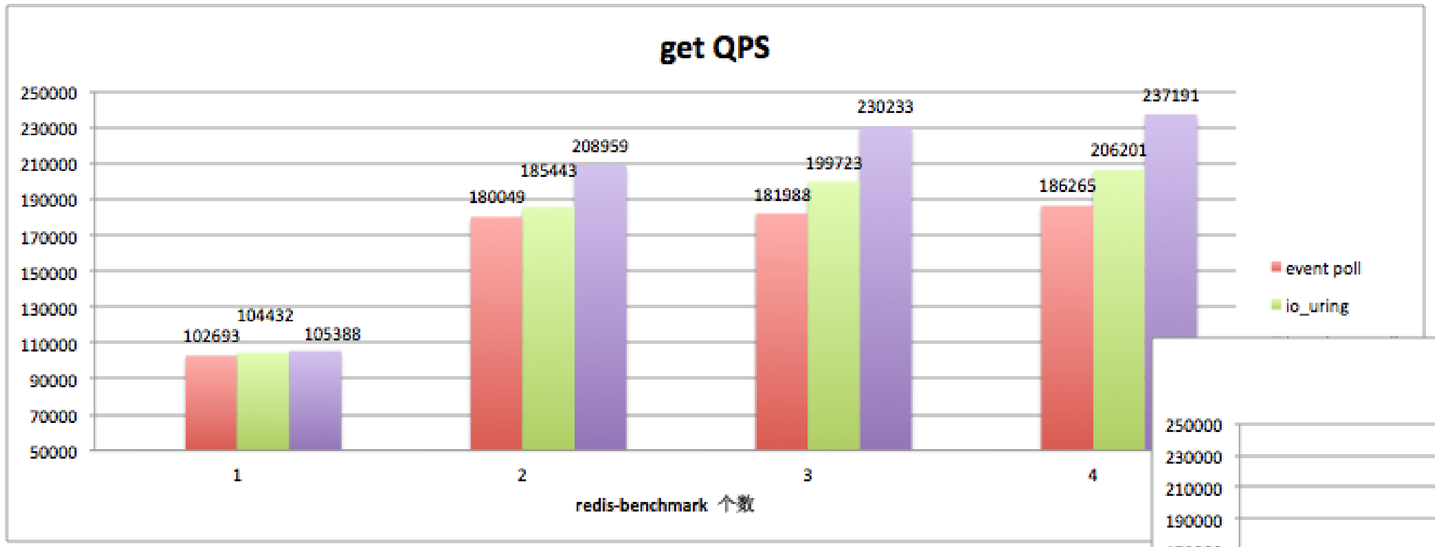


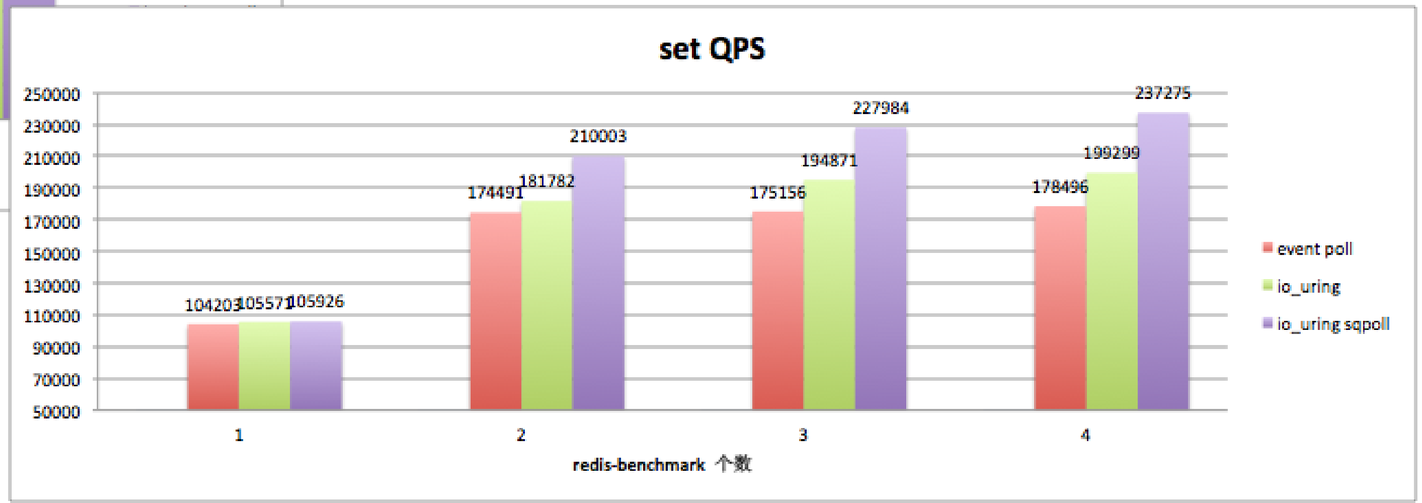


Redis

阿里云 | 奥运会全球指定云服务商

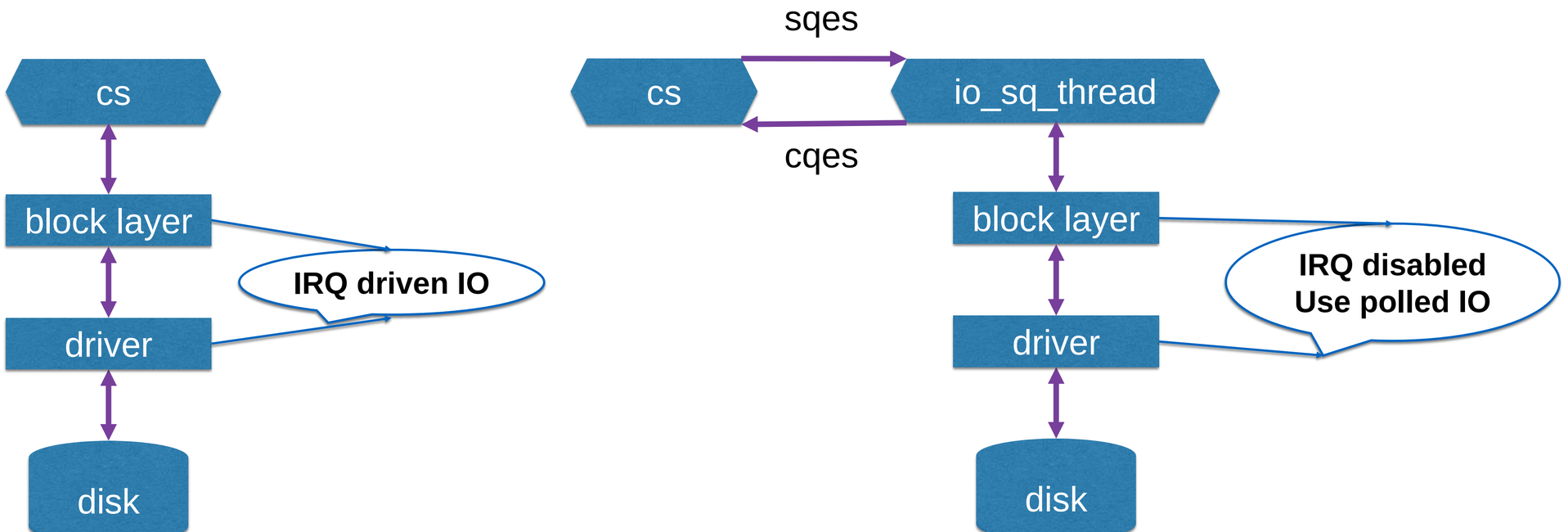
Redis 6.0 之前为单线程模型，客户端请求的处理都在主线程内完成，通过 ae 事件模型以及 IO 多路复用技术来高速的处理客户端请求。利用 io uring 的异步下发和 FAST POLL 机制优化之后, 高负载情况下，io_uring 相比 event poll，吞吐提升 8%~11%。





内部自研 DB

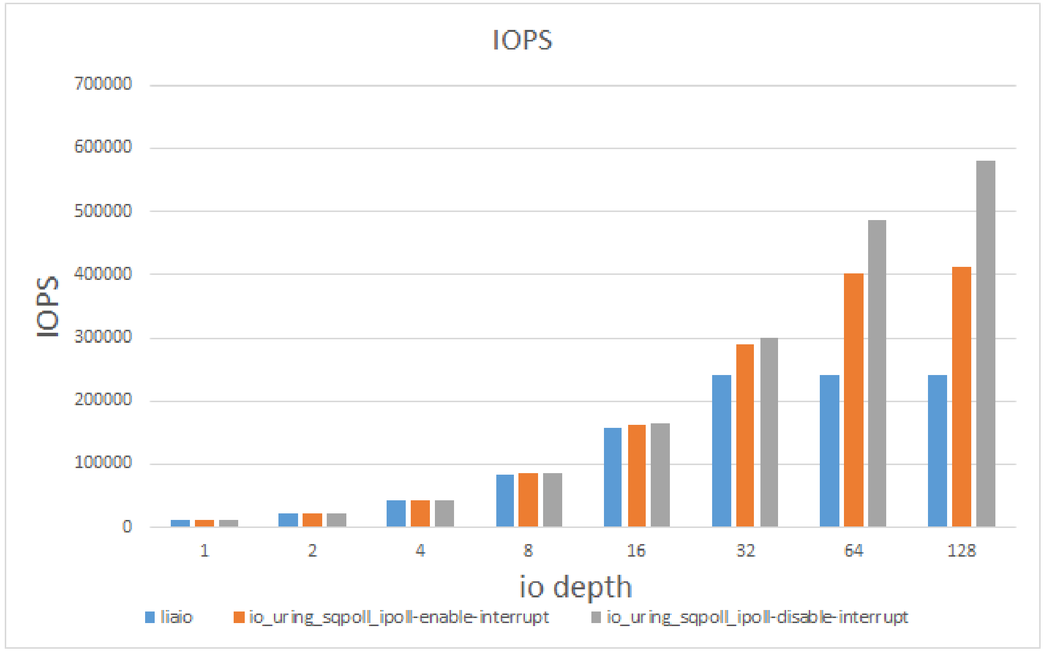
- 老架构的 cs 进程负责网络通信，数据处理，磁盘 IO 等工作，独占一个 CPU 核，采用 libaio 引擎。
- 新架构基于 io_uring，cs 进程利用 io_uring 的 SQPOLL & IOPOLL 特性，将磁盘 IO offload sqthread，单独绑定到一个新的 CPU 核。
- 利用 blk-mq 多队列映射机制，使 NVMe 盘预留一个 poll queue，且关闭该队列的硬件中断，IO 操作全部提交到该 poll queue，从而完整的利用内核 IO 栈 polling 特性。

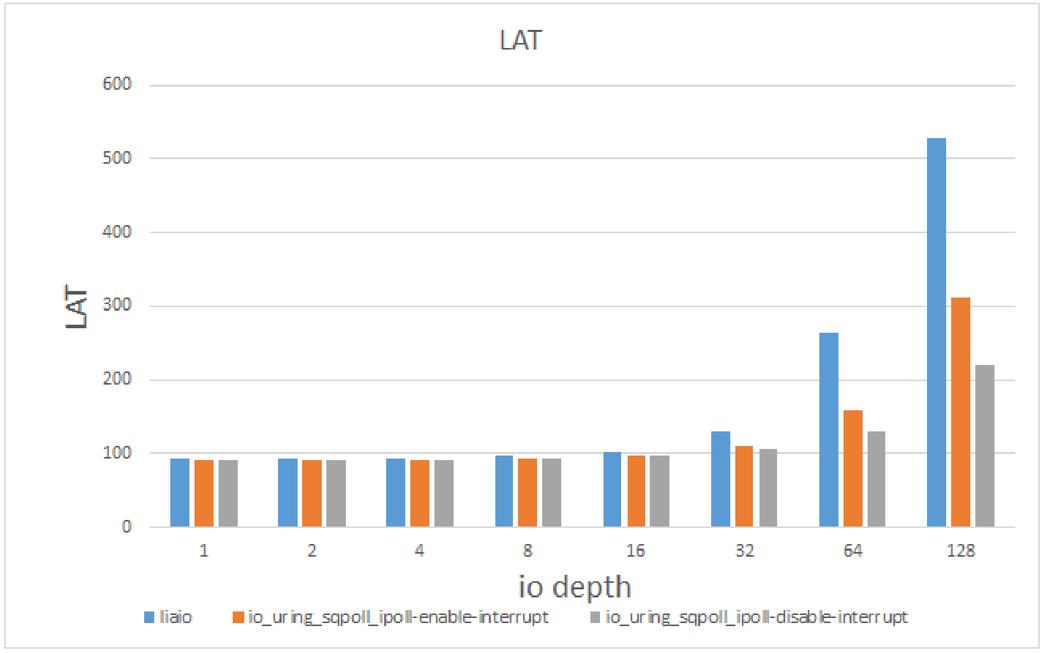


内部自研 DB 优化效果

阿里云 | 奥运会全球指定云服务商

- 三种模型的 IO 性能：libaio，io_uring 开启 sqpoll，io_uring 开启 sqpoll & iopoll。
- 在队列深度超过 32 时，iopoll 相比于中断方式带来显著的性能提升，且时延也保持在较低的水平。





Agenda

阿里云 | 奥运会全球指定云服务商

- 为什么需要 io_uring
- io_uring 架构及重要特性
- io_uring 优化工作
- io_uring 应用实践
- 下一步工作

下一步工作

- io_uring 框架 overhead 优化 & 稳定性

  - 高 IO 压力场景，perf 显示 io_uring 框架本身占据着接近 10% 的 overhead，存在较大优化空间。

  - iopoll 场景，IO 提交和完成都在一个线程上下文，大量同步操作可以去除，可进一步提高性能。

  - io_uring 社区高速发展，代码质量，稳定性等问题突出。

- block layer 性能优化

  - 借助 io_uring，可以更容易的发现内核 IO 栈软件瓶颈。

  - io stat，merge 以及 plug 都会对性能造成影响。

  - 借助 iopoll 进一步的对 Block 层进行优化，真正在内核提供一个比肩 SPDK 的 IO 栈。

- io_uring 生态

  - 越来越多的应用开始适配 io_uring

  - 不断出现新的基于 io_uring 构建出来高性能解决方案

相关工作的文档和代码已经 openanolis.org 开源，欢迎参与进来一起学习和探索！

- https://openanolis.org/
- https://github.com/openanolis

阿里云 | 奥运会全球指定云服务商

Q & A

奥运会全球指定云服务商

WWW.ALIYUN.COM