io_uring
Efficient ASync I/O on Modern Linux

Chris McFarlen, Mo Chen

Before io_uring

- Linux AIO has problems
  - Not truly async
  - Bouncing buffered IO to another thread
    - What if it's actually cached?
- syscalls are expensive
  - Spectre/meltdown mitigations have made it worse

AIO is a horrible ad-hoc design, with the main excuse being "other, less gifted people, made that design, and we are implementing it for compatibility because database people — who seldom have any shred of taste — actually use it".

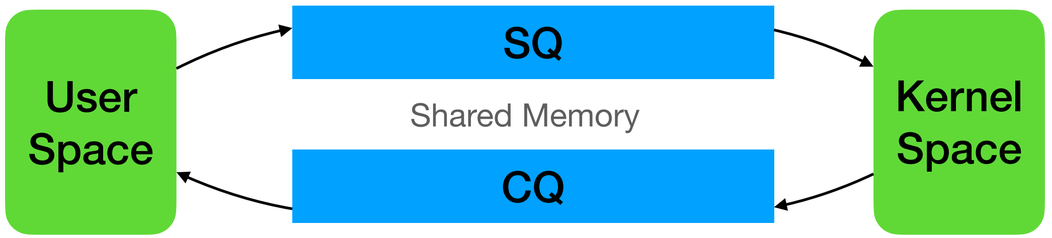
— Linus Torvalds (on lwn.net)

What is io_uring?

- Communication channel
- Lots of operations
  - Block I/O
  - File I/O
  - Network I/O
  - etc.
- Uniform async programming interface

io_uring interface

- Submission queue
- Completion queue
- single producer single consumer queues
  - lock free
  - shared memory between user and kernel
  - Each io_uring instance is typically single threaded



New syscalls

- int io_uring_setup(u32 entries, struct io_uring_params *p)
- int io_uring_enter(unsigned int fd, unsigned int to_submit, unsigned int min_complete, unsigned int flags, sigset_t *sig)

```
1 struct io_uring_sqe *sqe;
2 unsigned tail, index;
3 tail = *sqring->tail;
4 index = tail & (*sqring->ring_mask);
5 sqe = &sqring->sqes[index];
6 /* fill up details about this I/O request */
7 describe_io(sqe);
8 /* fill the sqe index into the SQ ring array */
9 sqring->array[index] = index;
10 tail++;
11 atomic_store_release(sqring->tail, tail);
```

Submit work

```
1 unsigned head;
2 head = *cqring->head;
3 if (head != atomic_load_acquire(cqring->tail)) {
4     struct io_uring_cqe *cqe;
5     unsigned index;
6     index = head & (cqring->mask);
7     cqe = &cqring->cqes[index];
8     /* process completed CQE */
9     process_cqe(cqe);
10     /* CQE consumption complete */
11     head++;
12 }
13 atomic_store_release(cqring->head, head);
```

Read completion events

liburing

- Userspace library
- Easy to use

```
1 struct io_uring ring;
2 ret = io_uring_queue_init(1, &ring, 0);
3 if (ret) {
4         fprintf(stderr, "queue init failed: %d\n", ret);
5         return 1;
6 }
7
8 io_uring_sqe* cqe;
9 io_uring_sqe* sqe = io_uring_get_sqe(&ring);
10 io_uring_prep_send(sqe, sockfd, data, len, 0);
11 sqe->user_data = 1;
12
13 ret = io_uring_submit(&ring);
14 if (ret <= 0) {
15         fprintf(stderr, "submit failed: %d\n", ret);
16         return 1;
17 }
18
19 ret = io_uring_wait_cqe(&ring, &cqe);
20 if (cqe->res != len) {
21         fprintf(stderr, "failed cqe: %d\n", cqe->res);
22         return 1;
23 }
```

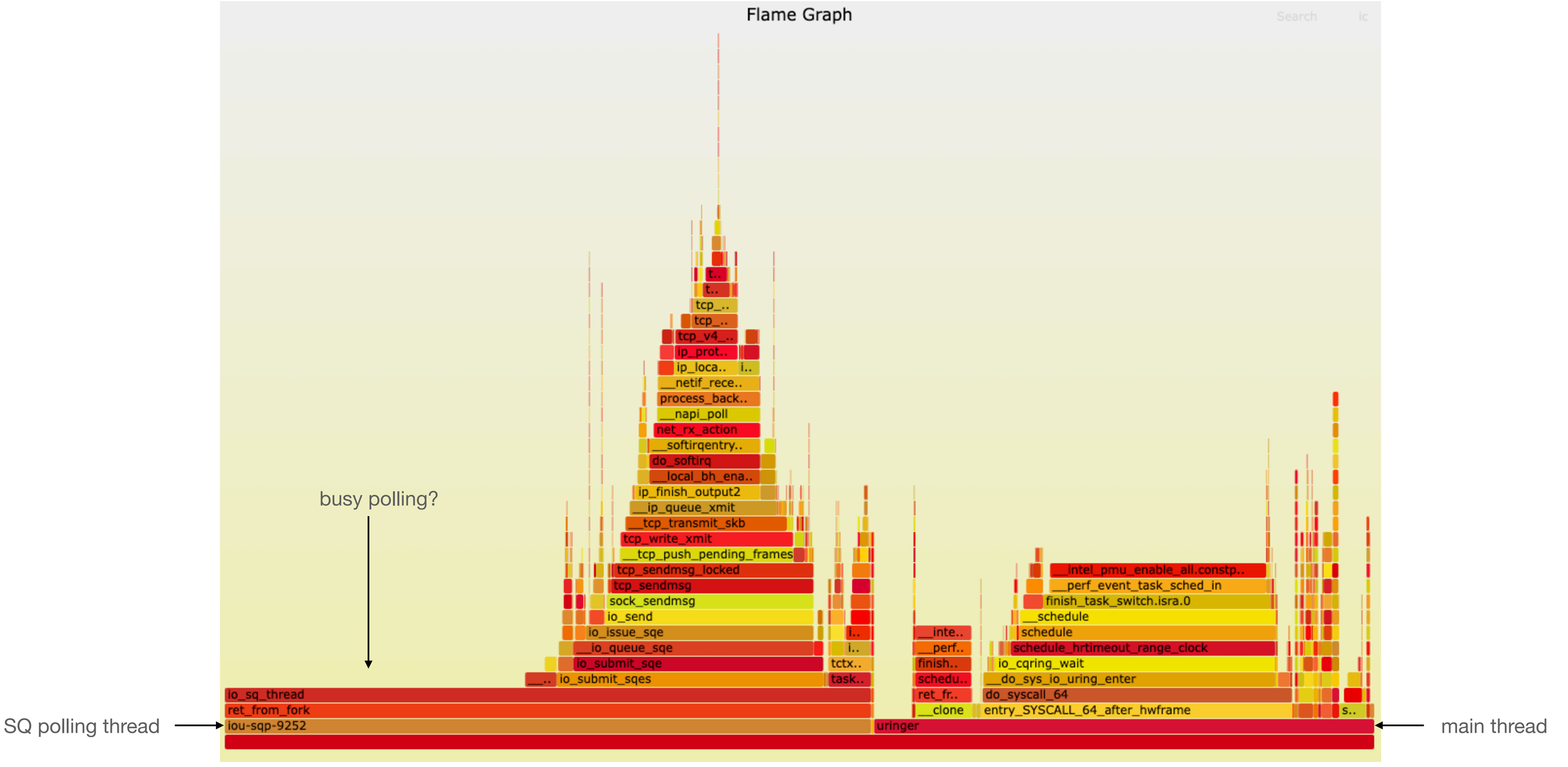
Features

Polling

- IORING_SETUP_SQPOLL
  - Submission queue polling
- busy polling completions
  - Get completed operation results
- IORING_SETUP_IOPOLL
  - Perform busy-waiting for an I/O completion, as opposed to getting notifications via an asynchronous IRQ (Interrupt Request).
  - lower latency busy polling for supported file system or block devices


Flame Graph
do_softirq (6,946 samples, 34.70%)
Search
ic
i..
_..
__w..
__w..
sock..
sk..
__..
tcp_clean_rt..
tcp_ack
tcp_rcv_established
tcp_v4_do_rcv
i..
tcp_v4_rcv
i..
nf..
ip_protocol_deliver_rcu
nf_..
ip_..
ip_local_deliver_finish
ip_rcv
__netif_receive_skb_one_core
process_backlog
__napi_poll
net_rx_action
x..
__softirqentry_text_start
ipt..
d..
do_softirq
nf_ho..
__de..
__local_bh_enable_ip
__ip_l..
ip_finish_output2
ip_loc..
__ip_queue_xmit
__tcp_transmit_skb
tcp_write_xmit
__all..
tcp_r..
__tcp_push_pending_frames
sk_st..
tcp_rec..
tc..
tcp_sendmsg_locked
inet_re..
tcp_..
tcp_sendmsg
io_recv
inet..
sock_sendmsg
io_issue_..
_..
io_recv
io_send
__io_queu..
io..
io_issue_sqe
tctx_task_work
__io_queue_sqe
task_work_run
io_submit_sqe
exit_to_user_..
io_submit_sqes
exit_to_user_..
__do_sys_io_uring_enter
syscall_exit_to..
do_syscall_64
entry_SYSCALL_64_after_hwframe
uringer

main thread


Flame Graph
busy polling?
SQ polling thread
main thread
io_sq_thread
ret_from_fork
iou-sqp-9252
uringer
io_submit_sqes
io_submit_sqe
__io_queue_sqe
io_issue_sqe
io_send
sock_sendmsg
tcp_sendmsg
tcp_sendmsg_locked
__tcp_push_pending_frames
tcp_write_xmit
__tcp_transmit_skb
__ip_queue_xmit
ip_finish_output2
do_softirq
net_rx_action
__napi_poll
process_back..
__netif_rece..
ip_loca..
ip_prot..
tcp_v4_..
tcp_..
t..
__clone
ret_fr..
schedu..
finish..
__perf..
__inte..
entry_SYSCALL_64_after_hwframe
do_syscall_64
__do_sys_io_uring_enter
io_cqring_wait
schedule_hrtimeout_range_clock
schedule
__schedule
finish_task_switch.isra.0
__perf_event_task_sched_in
__intel_pmu_enable_all.constp..

Registered Resources

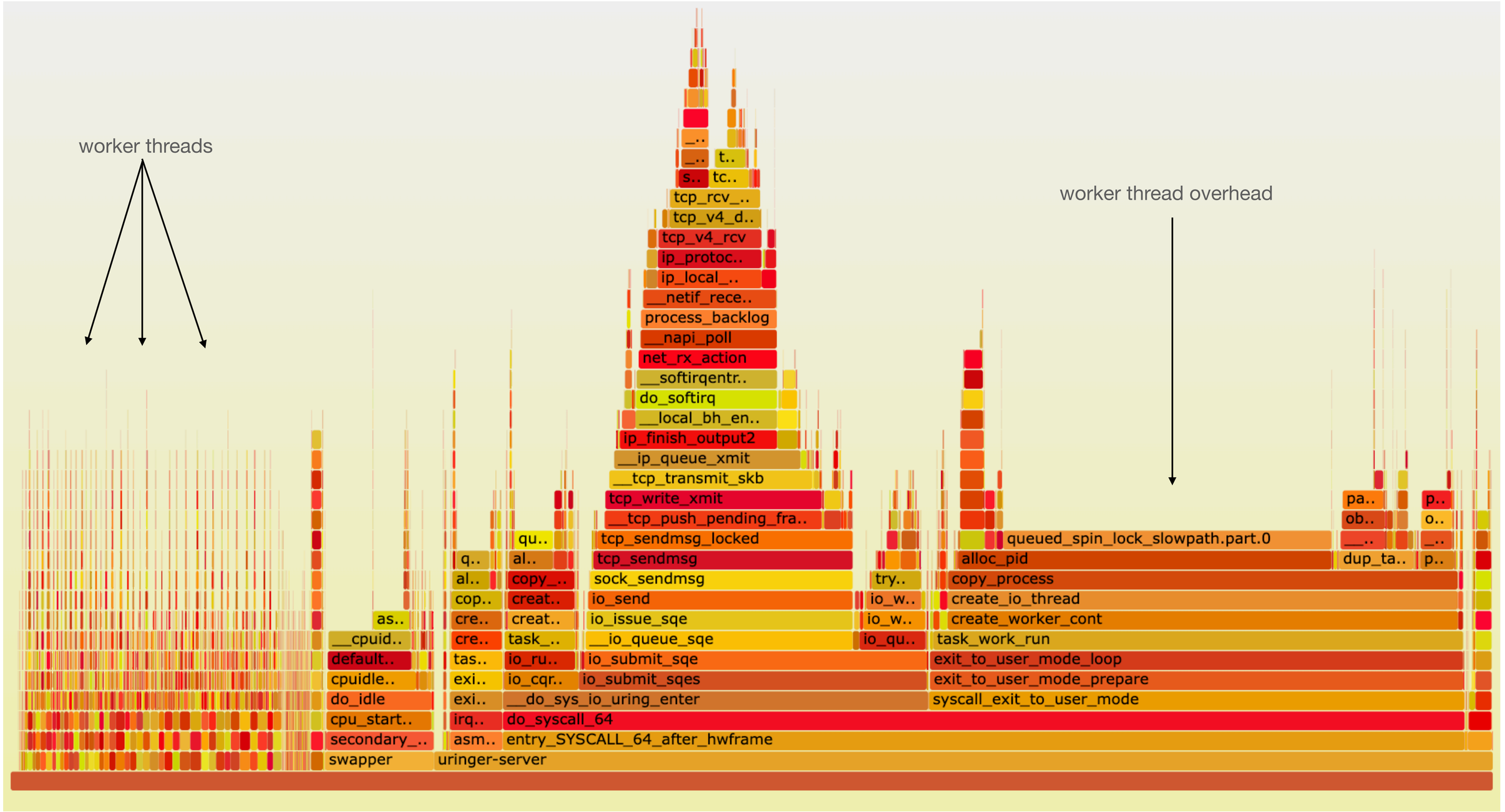
- Often-used resources
  - buffers
  - files
- Lower overhead

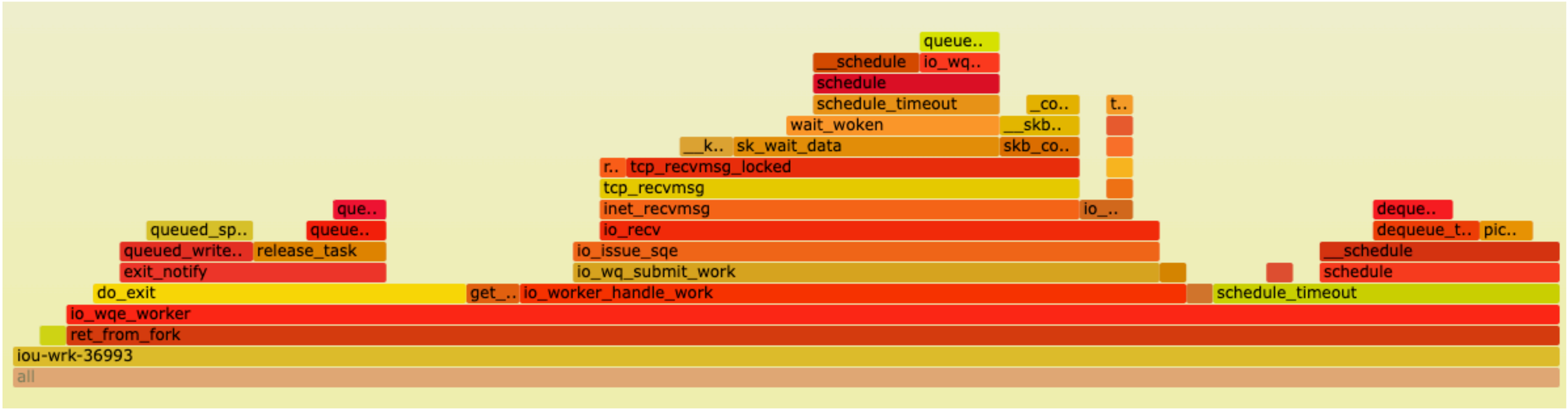
Other Features

- Linked operations
- Timeouts
- Cancellation
- eventfd
- CPU affinity

Internals

- How does io_uring actually perform I/O?
  - bounded vs unbounded
  - immediate
  - poll set
  - worker pool


worker threads
worker thread overhead
_..
_..
t..
s..
tc..
tcp_rcv_..
tcp_v4_d..
tcp_v4_rcv
ip_protoc..
ip_local_..
__netif_rece..
process_backlog
__napi_poll
net_rx_action
__softirqentr..
do_softirq
__local_bh_en..
ip_finish_output2
__ip_queue_xmit
__tcp_transmit_skb
tcp_write_xmit
__tcp_push_pending_fra..
qu..
tcp_sendmsg_locked
q..
al..
tcp_sendmsg
al..
copy_..
sock_sendmsg
cop..
creat..
io_send
cre..
creat..
io_issue_sqe
cre..
task_..
__io_queue_sqe
tas..
io_ru..
io_submit_sqe
exi..
io_cqr..
io_submit_sqes
exi..
__do_sys_io_uring_enter
irq..
do_syscall_64
asm..
entry_SYSCALL_64_after_hwframe
try..
io_w..
io_w..
io_qu..
queued_spin_lock_slowpath.part.0
alloc_pid
copy_process
create_io_thread
create_worker_cont
task_work_run
exit_to_user_mode_loop
exit_to_user_mode_prepare
syscall_exit_to_user_mode
pa..
p..
ob..
o..
__..
_..
dup_ta..
p..
as..
__cpuid..
default..
cpuidle..
do_idle
cpu_start..
secondary_..
swapper
uringer-server


queue..
__schedule
io_wq..
schedule
schedule_timeout
_co..
t..
wait_woken
__skb..
__k..
sk_wait_data
skb_co..
r..
tcp_recvmsg_locked
tcp_recvmsg
inet_recvmsg
io_..
que..
deque..
queued_sp..
queue..
io_recv
dequeue_t..
pic..
queued_write..
release_task
io_issue_sqe
__schedule
exit_notify
io_wq_submit_work
schedule
do_exit
get_..
io_worker_handle_work
schedule_timeout
io_wqe_worker
ret_from_fork
iou-wrk-36993
all

C++ coroutines

```
1       auto op = co_await r;
2
3       int rc = co_await op.open(path, dfd, O_CREATIO_TRUNCIO_WRONLY);
4       if (rc > 0) {
5           int fd = rc;
6           auto wr = co_await r;
7
8           rc = co_await wr.write(fd, content);
9
10           co_await (co_await r).close(fd);
11      }
12       co_return rc;
```

Performance

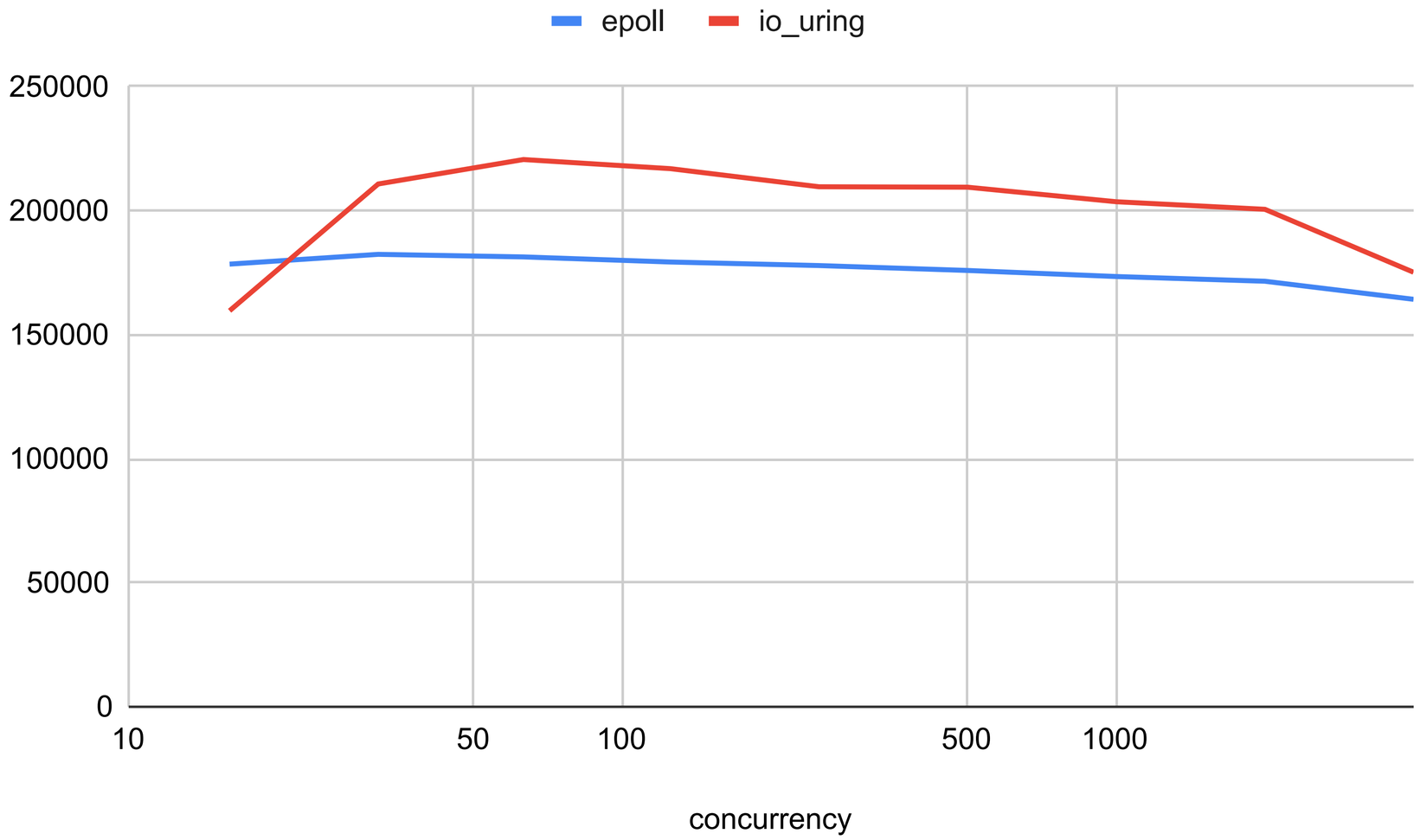
Disk I/O performance

backend	IOPS	context switches	IOPS ±% vs io_uring
sync	814,000	27,625,004	-42.6%
posix-aio (thread pool)	433,000	64,112,335	-69.4%
linux-aio	1,322,000	10,114,149	-6.7%
io_uring (basic)	1,417,000	11,309,574	—
io_uring (enhanced)	1,486,000	11,483,468	4.9%

From: https://thenewstack.io/how-io_uring-and-ebpf-will-revolutionize-programming-in-linux/

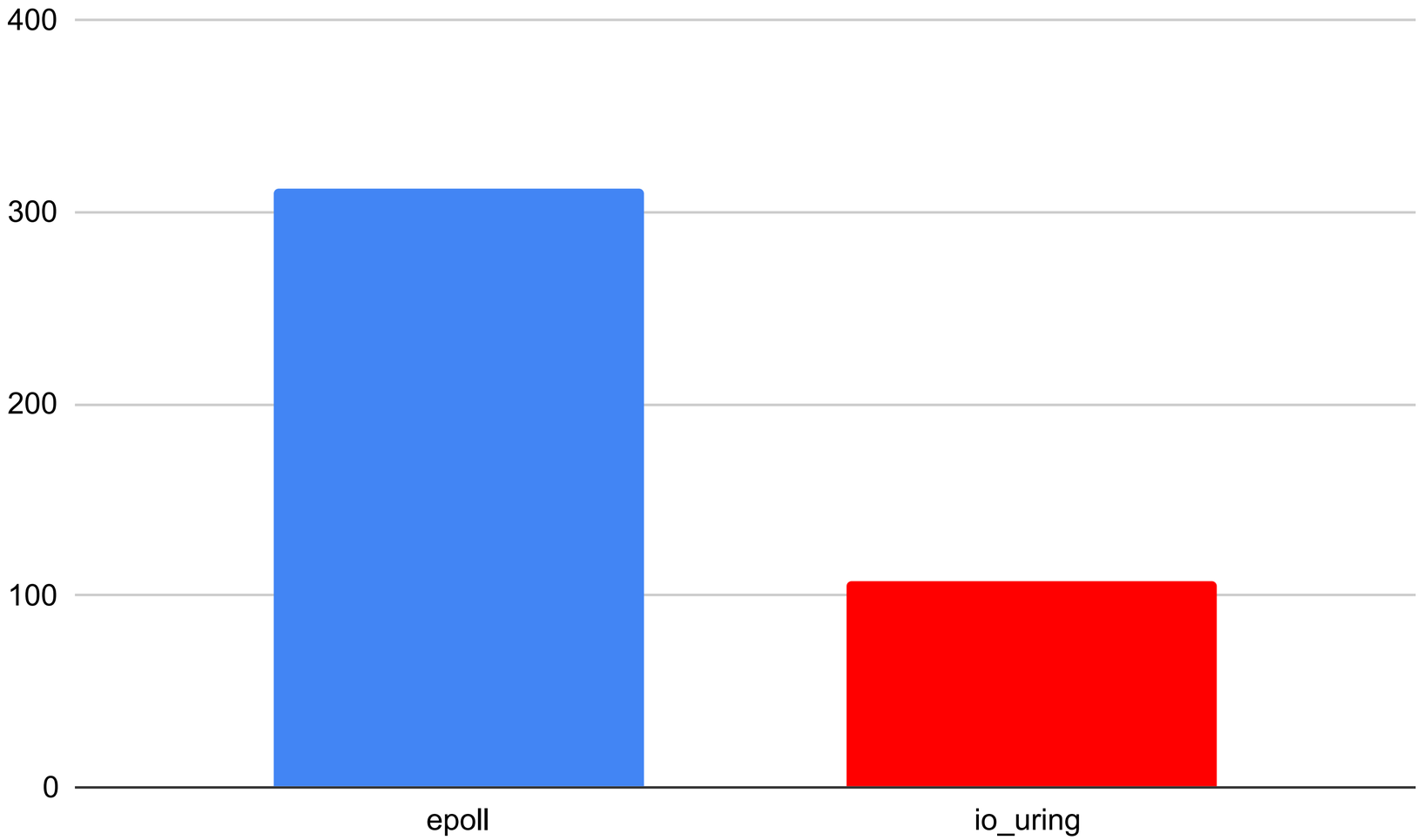
Echo server

throughput



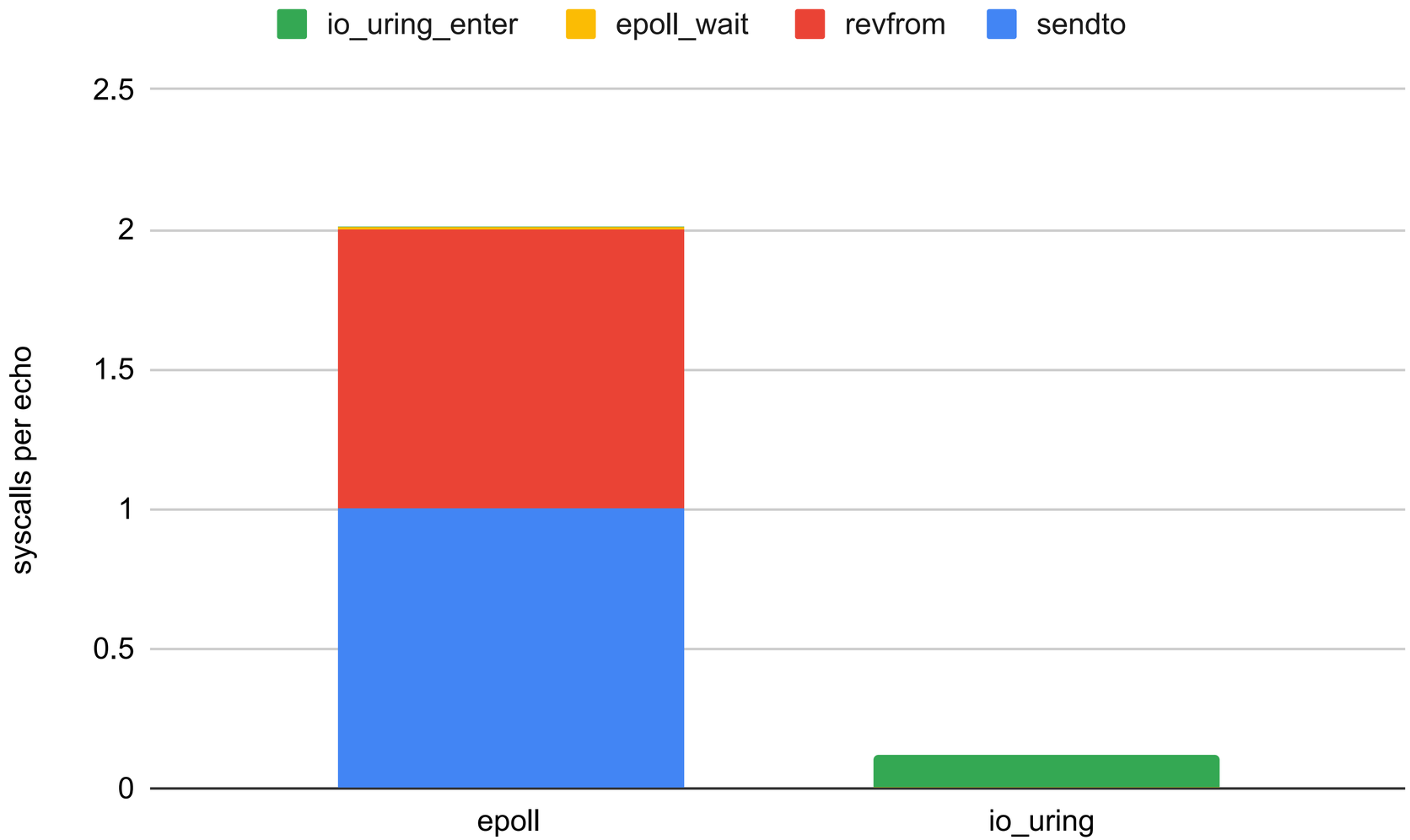
Echo server, c=1000

cycles per echo

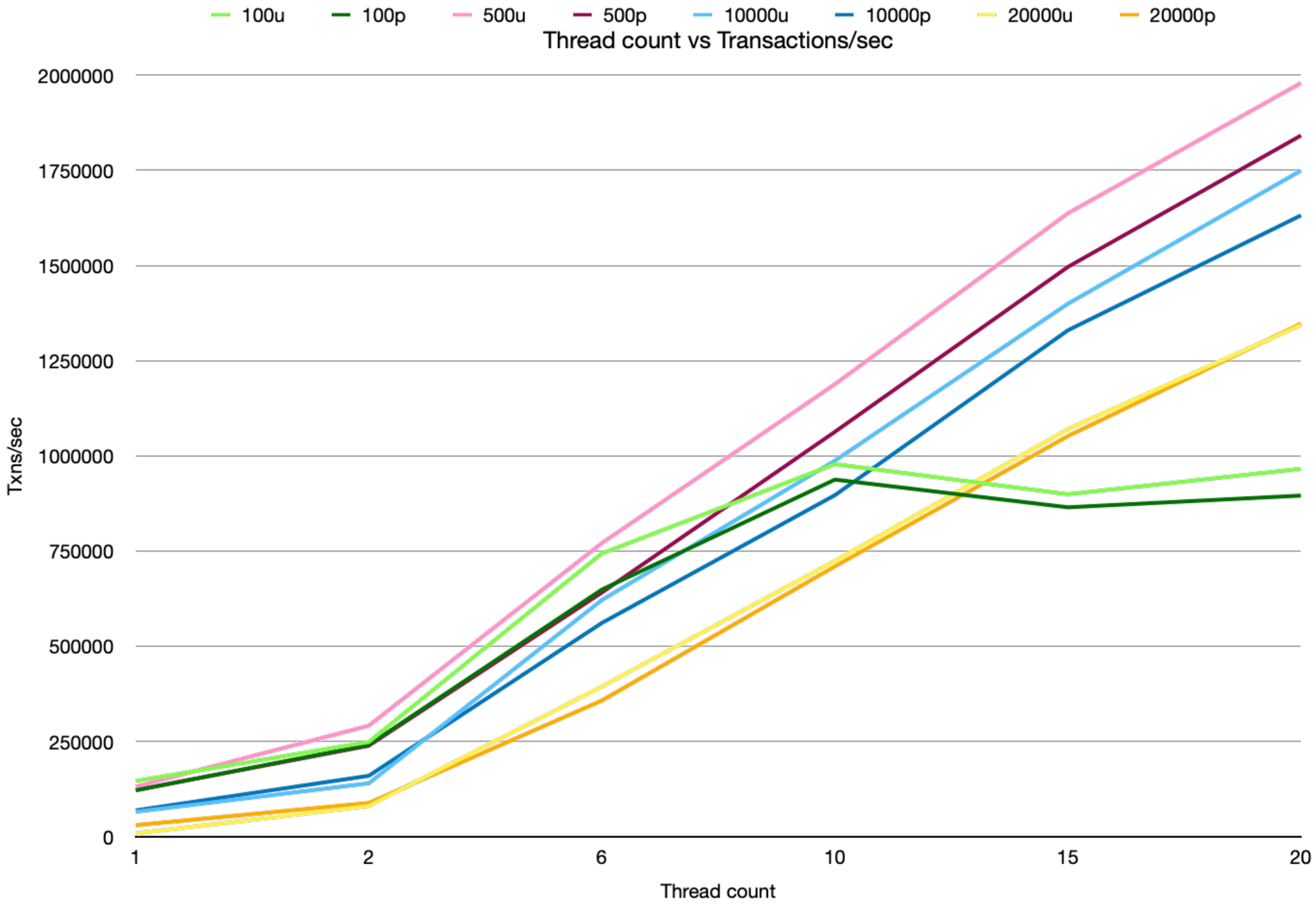


Echo server, c=1000

syscall count per echo



Client performance - io_uring vs poll



Debugging io_uring

- probes
  - kprobe - kernel addresses
  - tracepoint - kernel static tracepoints
- eBPF
  - run sandboxed programs in the kernel
- bpftrace

```
$ sudo bpftrace --btf -e 'kr:create_io_thread { @[retval] = count(); } i:s:1 { print(@); clear(@); } END { clear(@); }' -c '/usr/bin/sleep 3' | cat -s
Attaching 3 probes...
@[-11]: 293631
@[-11]: 306150
@[-11]: 311959
```

io_uring development is ongoing