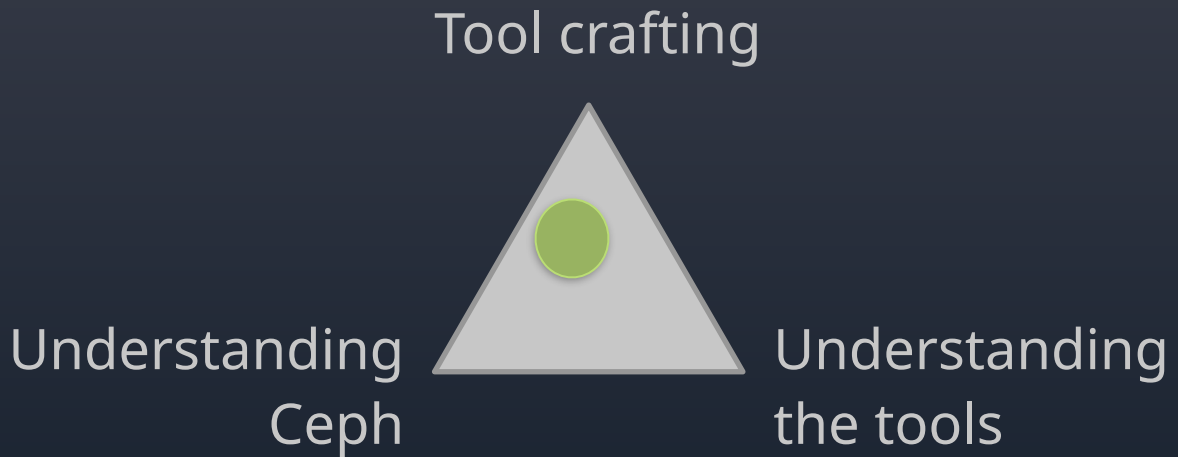
Understanding Ceph

A Journey from Metrics to Tracing

Marcel Lauhoff | Staff Software Engineer
February, 2025

Scope & Goals

Sequel to *Understanding Ceph - One Performance Counter at a Time (Cephalocon 2024)*



Let’s develop an intuition!

CL:SO

TOC

Intro: Ceph Metrics, Tracing

EX 1: Counters → Event Tracing

EX 2: Latency Metrics → Tracing for Metrics

Definitions

A **metric** is a **measurement** of a service captured at runtime (OpenTelemetry)

Examples:

librados sends a write operation → increment osdop_write counter

OSD processed an operation → update op_latency

Tracing [...] refers to the process of capturing and recording information about the execution of a software program. (Wikipedia)

Examples:

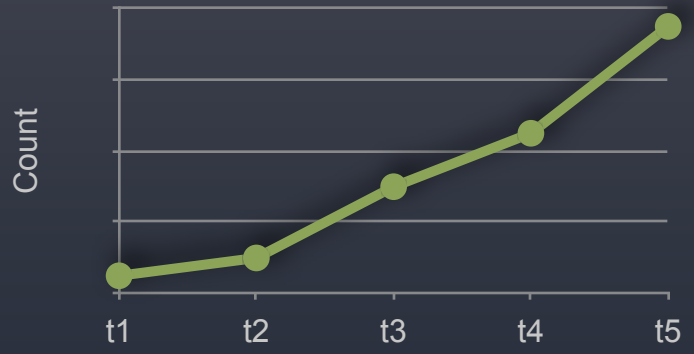
Record all RADOS operations

What function issues all the watch operations?

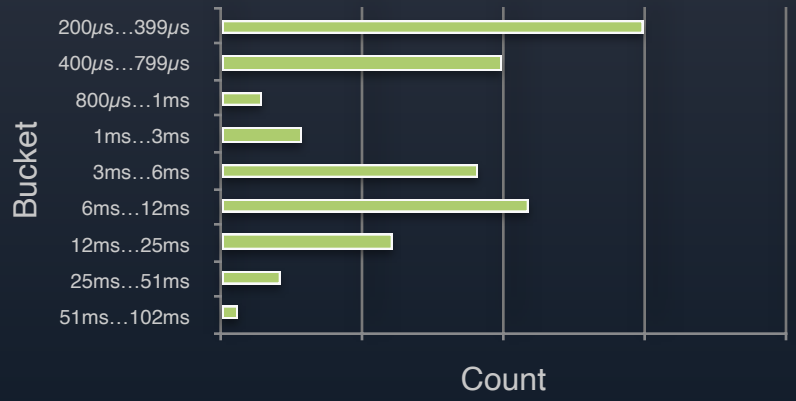
Capturing all open() or exec() on a system

Gauge

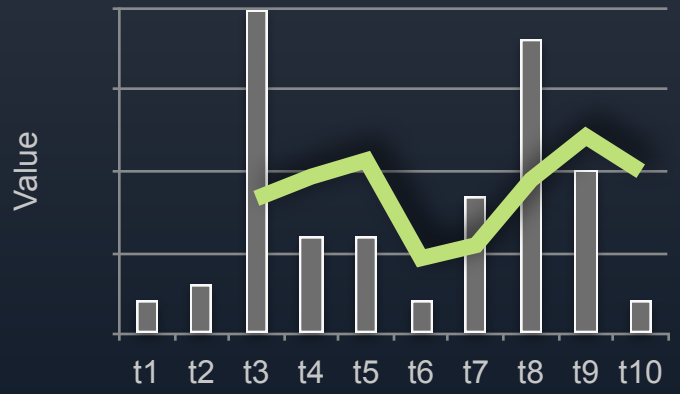
Counter



Histogram



Long Running Average



Ceph Tracing Options

uprobes

LLTng

USDT

Blkin

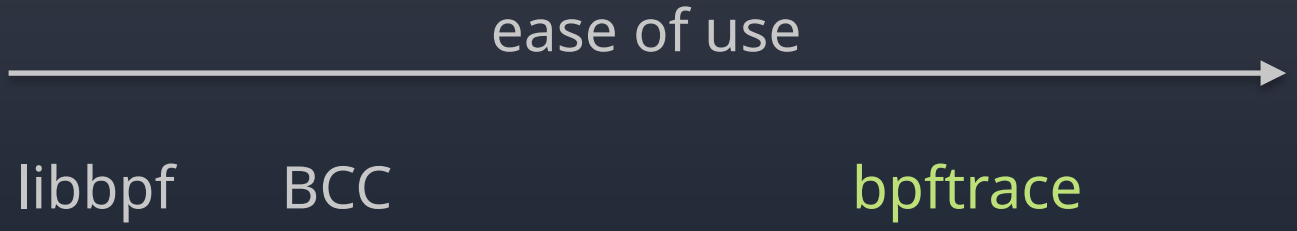
Zipkin

OpenTelemetry

Tracing

“Jaeger Tracing”

eBPF Tracing Frontends



Ceph LTTng Probes

`tracepoint(osdc, objecter_finish_op, op->tid, op->target.osd)`

≈≈ `_SDT_PROBE()` (USDT) + lttng_ust_do_tracepoint

≈≈ `_SDT_PROBE:` NOP + ELF metadata

→ We can bpftrace Ceph's LTTng tracepoints

What does this trace eBPF thing do anyway?

It runs **your** tracing code

→ Generate output

→ Count events

→ Condense measurements into histograms, stats, …

(eBPF) Tracing Challenges

☹️ Access to (deeply nested) data structures and C++ STL

😐 debuginfo helps, but not practical

🙄 “mocking” structs with placeholders

😵 eBPF stack limitations and strings

😀 USDT tracepoints with “extracted” arguments

⚠️ librados tracepoints not hit by RGW

EX 1: Counters → Event Tracing

Single 4MB S3 PUT to RADOS Gateway

```
2024-11-29T12:42:45.821+0100
  7f0be538f6c0  1
  beast: 0x7f0c53e61200: ::1 - testid
  [29/Nov/2024:12:42:45.758 +0100]
  "PUT /testbucket/13359 HTTP/1.1"
  200 4194304 - - - latency=0.063001677s
```

libRADOS Object Counter

objecter.{osdop|omap}

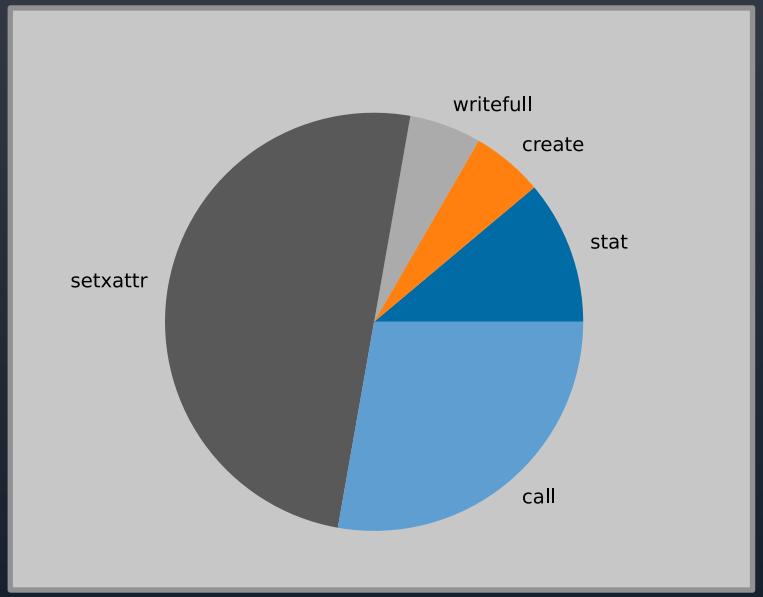
Table 1

"osdop_stat"	2
"osdop_create"	1
"osdop_read"	0
"osdop_write"	0
"osdop_writefull"	1
"osdop_writesame"	0
"osdop_append"	0
"osdop_zero"	0
"osdop_truncate"	0
"osdop_delete"	0
"osdop_mapext"	0
"osdop_sparse_read"	0
"osdop_clonerange"	0
"osdop_getxattr"	0
"osdop_setxattr"	9
"osdop_cmpxattr"	0
"osdop_rmxattr"	0
"osdop_resetxattrs"	0
"osdop_call"	5
"osdop_watch"	0
"osdop_notify"	0
"osdop_src_cmpxattr"	0
"osdop_pgls"	0
"osdop_pgls_filter"	0
"osdop_other"	0
“omap_wr"	0
“omap_rd"	0
“omap_del"	0

libRADOS Object Counter

objecter.{osdop|omap}_*

stat	2
create	1
writefull	1
setxattr	9
call	5



$\sum \ldots = 18$

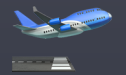
RGW: Single 4MB S3 PUT

∑ (osdop_*, omap_*) = 18

∑ msgr_send_messages = 3

objecter.op = 3

CL'SO

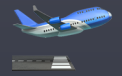


```
.dir.903d2ae0-8d7f-4edc-b65d-e3abe1732c23.4484.2.7
  stat
  → call rgw.guard_bucket_resharding in=36b
  → call rgw.bucket_prepare_op in=217b

903d2ae0-8d7f-4edc-b65d-e3abe1732c23.4484.2_28759
  create
  → setxattr user.rgw.idtag (62) in=76b
  → setxattr user.rgw.tail_tag (62) in=79b
  → writefull 0~4194304 in=4194304b
  → setxattr user.rgw.manifest (351) in=368b
  → setxattr user.rgw.acl (147) in=159b
  → setxattr user.rgw.content_type (25) in=46b
  → setxattr user.rgw.etag (32) in=45b
  → setxattr user.rgw.x-amz-meta-s3cmd-attrs (139) in=170b
  → call rgw.obj_store_pg_ver in=44b
  → setxattr user.rgw.source_zone (4) in=24b
  → setxattr user.rgw.storage_class (8) in=30b

.dir.903d2ae0-8d7f-4edc-b65d-e3abe1732c23.4484.2.7
  stat
  → call rgw.guard_bucket_resharding in=36b
  → call rgw.bucket_complete_op in=374b
```

radostrace (bpftrace version)



```
BEGIN {
  printf("%1s %-9s %4s %-32s %s\n", "⇆", "TID", "T(ms)", "OBJ", "OPS");
}

usdt:…/libceph-common.so:osdc:objecter_send_op {
   printf("%1s %-9d %4s %-32s %s\n",
          "→", arg0, "", str(arg2), str(arg4));
   @start[arg0] = nsecs;
}
usdt:…/libceph-common.so:osdc:objecter_finish_op /@start[arg0]/ {
   $duration_ms = (nsecs - @start[arg0])/1000000;
   printf("%1s %-9d %4d\n", "←", arg0, $duration_ms);
   delete(@start[arg0]);
}
```

What does this teach us?

Op Mix

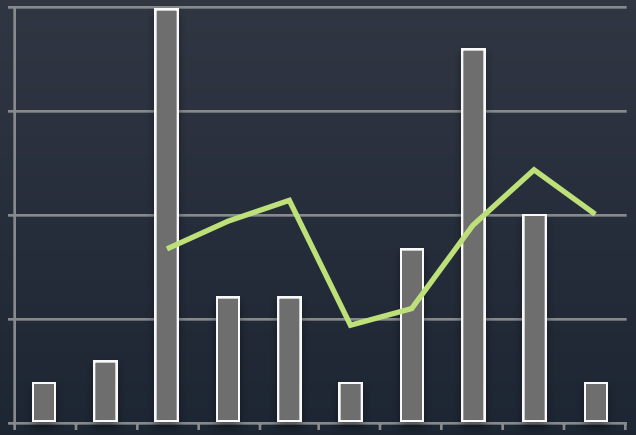
OP = [op, ..., op]

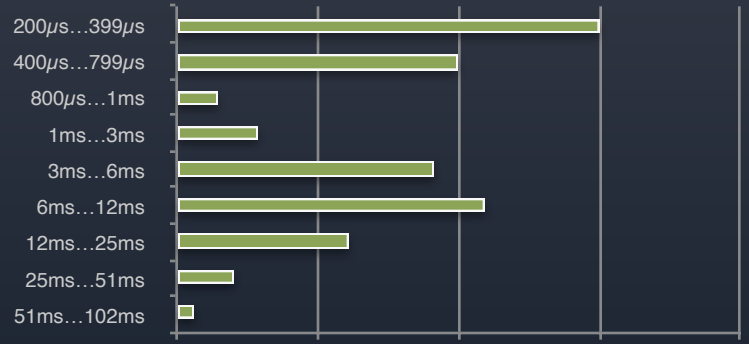
3 messages, 3 OPS, 18 ops

→ 2 OPS Bucket Index Transaction, 1 Object + Metadata

CL'SO

EX 2: Latency Metrics → Tracing for Metrics

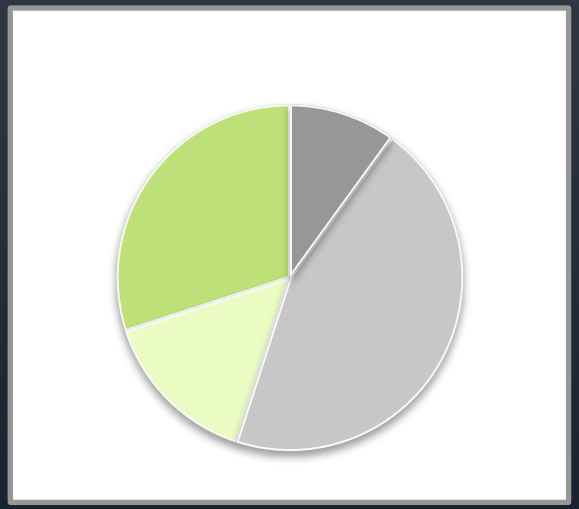


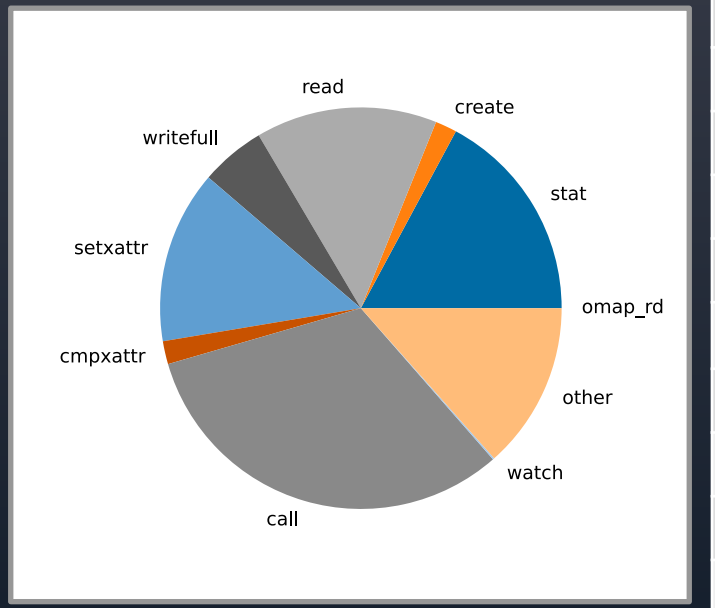


CL:SO

S3 Mixed Workload Benchmark

10% DELETE, 45% GET, 15% PUT, 30% STAT





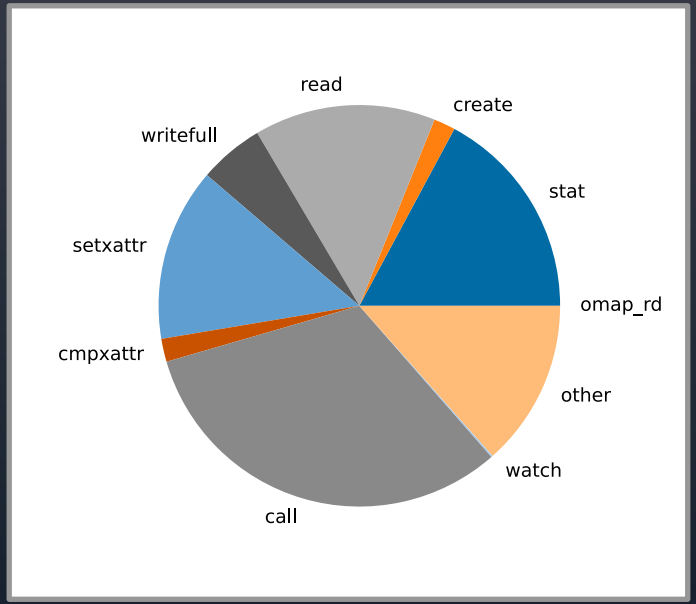
stat	20947
create	2125
read	17836
writefull	6375
setxattr	17000
cmpxattr	2265
call	39004
watch	150
other	16417
omaprd	2

S3 Mixed Workload Benchmark

osd.op_latency

~40 ms average latency

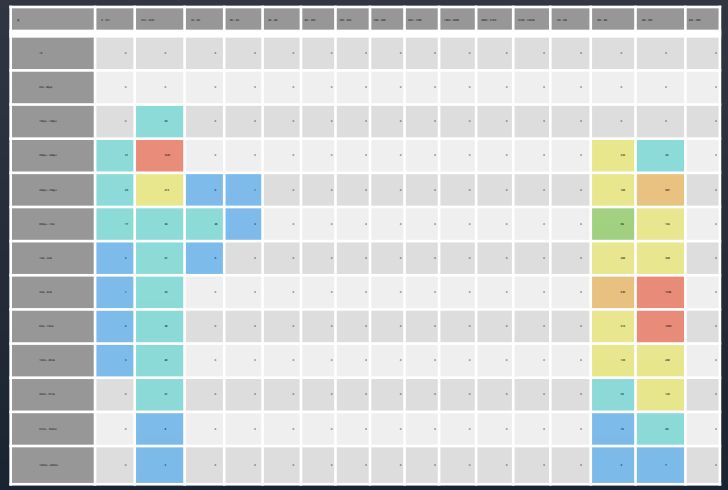
Is this actually meaningful?



CL'SO

More detail?

More averages..



Perf Counter Histograms

Tracing

Events

Counter

Histograms

OSD Op Tracepoints

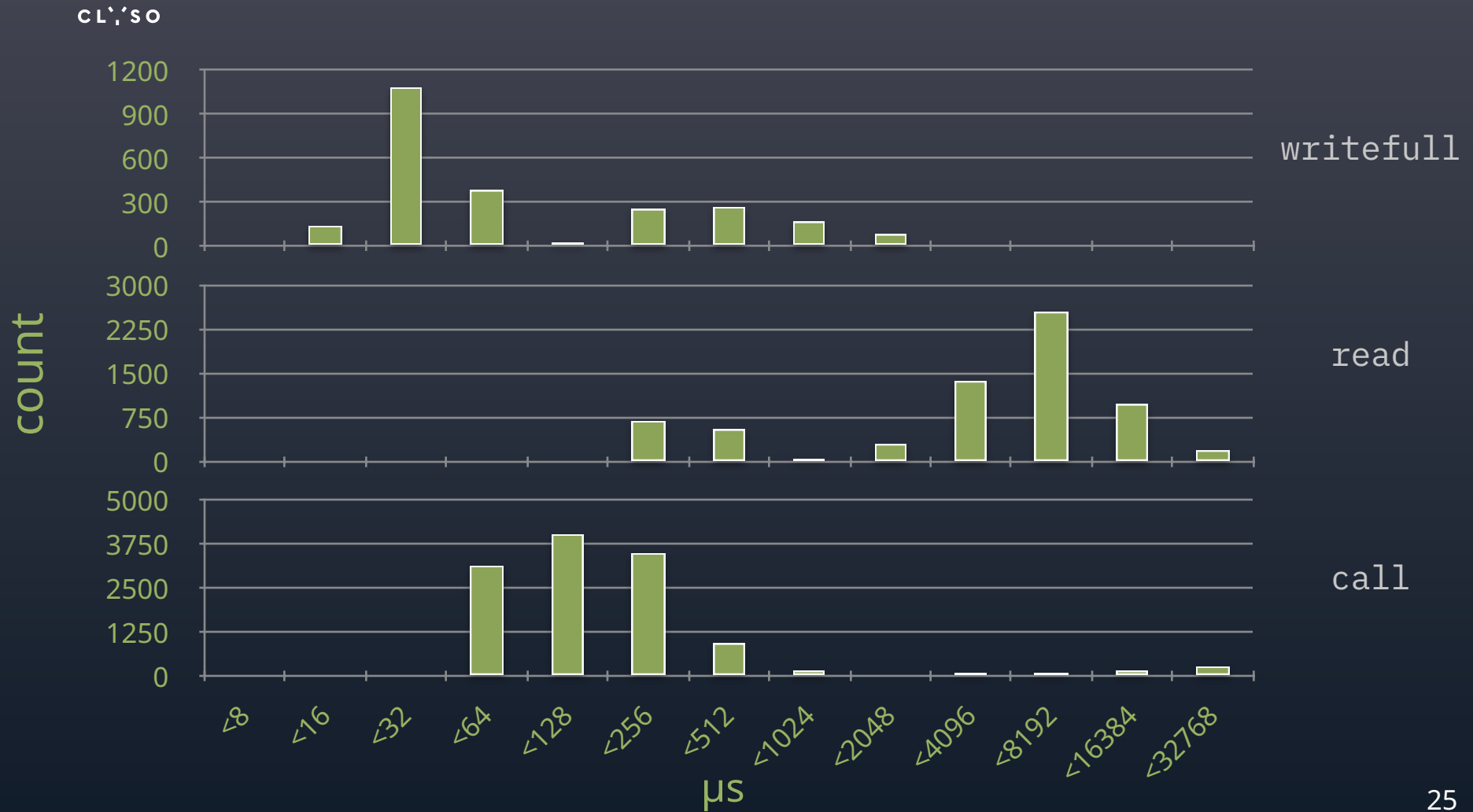
do_osd_op_pre → do_osd_op_pre_* → do_osd_op_post

do_osd_op_pre_append
do_osd_op_pre_assert_ver
do_osd_op_pre_cache_evict
do_osd_op_pre_cache_flush
do_osd_op_pre_cache_pin
do_osd_op_pre_cache_unpin
do_osd_op_pre_call
do_osd_op_pre_checksum
do_osd_op_pre_cmpxattr
do_osd_op_pre_copy_from
do_osd_op_pre_copy_get
do_osd_op_pre_create
do_osd_op_pre_delete
do_osd_op_pre_extent_cmp
do_osd_op_pre_getxattr
do_osd_op_pre_getxattrs
do_osd_op_pre_isdirty
do_osd_op_pre_list_snaps
do_osd_op_pre_list_watchers
do_osd_op_pre_mapext
do_osd_op_pre_notify
do_osd_op_pre_notify_ack
do_osd_op_pre_omap_cmp
do_osd_op_pre_omapclear
do_osd_op_pre_omapgetheader
do_osd_op_pre_omapgetkeys
do_osd_op_pre_omapgetvals
do_osd_op_pre_omapgetvalsbykeys
do_osd_op_pre_omaprmkeyrange
do_osd_op_pre_omaprmkeys
do_osd_op_pre_omapsetheader
do_osd_op_pre_omapsetvals
do_osd_op_pre_read
do_osd_op_pre_rmxattr
do_osd_op_pre_rollback
do_osd_op_pre_setallochint
do_osd_op_pre_setxattr
do_osd_op_pre_sparse_read
do_osd_op_pre_stat
do_osd_op_pre_tmap2omap
do_osd_op_pre_tmapget
do_osd_op_pre_tmapput
do_osd_op_pre_tmapup
do_osd_op_pre_truncate
do_osd_op_pre_try_flush
do_osd_op_pre_undirty
do_osd_op_pre_unknown
do_osd_op_pre_watch
do_osd_op_pre_write
do_osd_op_pre_writefull
do_osd_op_pre_writesame
do_osd_op_pre_zero

CL'SO

OSD: What is the most accessed object?

0	.dir.59ff0c8a-3962-4081-932e-dcaa497a565f.4482.5.6 .dir.59ff0c8a-3962-4081-932e-dcaa497a565f.4482.5.8 .dir.59ff0c8a-3962-4081-932e-dcaa497a565f.4482.5.4 gc.14 gc.15
1	.dir.59ff0c8a-3962-4081-932e-dcaa497a565f.4482.5.2 .dir.59ff0c8a-3962-4081-932e-dcaa497a565f.4482.5.3 .dir.59ff0c8a-3962-4081-932e-dcaa497a565f.4482.5.0 gc.11 gc.7
2	.dir.59ff0c8a-3962-4081-932e-dcaa497a565f.4482.5.9 .dir.59ff0c8a-3962-4081-932e-dcaa497a565f.4482.5.10 .dir.59ff0c8a-3962-4081-932e-dcaa497a565f.4482.5.1 .dir.59ff0c8a-3962-4081-932e-dcaa497a565f.4482.5.7 .dir.59ff0c8a-3962-4081-932e-dcaa497a565f.4482.5.5


writefull
read
call
count
1200
900
600
300
0
3000
2250
1500
750
0
5000
3750
2500
1250
0
<8
<16
<32
<64
<128
<256
<512
<1024
<2048
<4096
<8192
<16384
<32768
µs

CL:'SO

How?

```
usdt:*:osd:do_osd_op_post {
 @total[str(arg0)] = count();
}
```

```
usdt:*:osd:do_osd_op_pre {
  @start[tid] = nsecs;
}

usdt:*:osd:do_osd_op_post {
  $elapsed =
   (nsecs - @start[tid]) / 1000;

  @total_us[str(arg3)] =
   hist($elapsed);
}
```

CL'SO

Can we trace sth useful with just uprobes?

```
// RGWOp::complete(..)
uprobe:…/radosgw:_ZN5RGWOp8completeEv {
  $op = (RGWOp*)(arg0);
  $req_id = $op->s->id;
  $elapsed = (nsecs - @reqs[$req_id]) / 1000000;
  $opcode = $op->s->op_type;
  @lat_hist_ms[$opcode] = hist($elapsed);
  @lat_stat[$opcode] = stats($elapsed);
  delete(@reqs[$req_id])
}
// RGWOp::init(..)
uprobe:…/radosgw:_ZN5RGWOp4initEPN3rgw3sal6DriverEP9req_stateP10RGWHandler {
  $op = (RGWOp*)(arg0);
  $req = (struct req_state*)(arg2);
  $req_id = $req->id;
  @reqs[$req_id] = nsecs;
}
```

Recap

How to trace Ceph + Challenges

Counters → Event Tracing

We can trace for events,
but also aggregate them into metrics

What's next?

ebpf_exporter for Ceph?

Error tracing

A whatsup? tracer

Integrate with logger - conditional log messages

Let's build a Ceph BPF Tracing Toolkit!

https://github.com/clyso/ceph-ebpf-toolkit

https://github.com/irq0/talk_ceph_perf_counters

CL'SO
Powered by ceph

https://github.com/clyso/ceph-ebpf-toolkit

Thank you!

Marcel Lauhoff <marcel.lauhoff@clyso.com>