第三届 eBPF开发者大会
www.ebpftravel.com

从内核到APP：eBPF驱动的跨层性能分析

荣耀终端|OS Kernel Lab

王子成 / 王震 / 刘璐

HONOR

中国·西安

作者介绍

- 王子成：博士毕业于南京大学计算机系，研究兴趣集中在OS内核安全、内存，漏洞防御缓解等，相关研究工作发表在Usenix Security，NDSS，ACSAC，软件学报等学术会议和期刊，并曾在CLK'23, Linux Security Summit'23/24, Blackhat'23/24，Chinasys'24报告研究工作进展
- 王震：荣耀，研究兴趣集中在OS内核调度、可观测性
- 刘璐：荣耀，研究兴趣集中在终端设备OS基础性能维护和提升，AI4OS

问题：如何更好呈现eBPF获取的内核观测数据？

eBPF似乎只有观测的“超能力”，缺乏观测数据呈现的“超能力”？

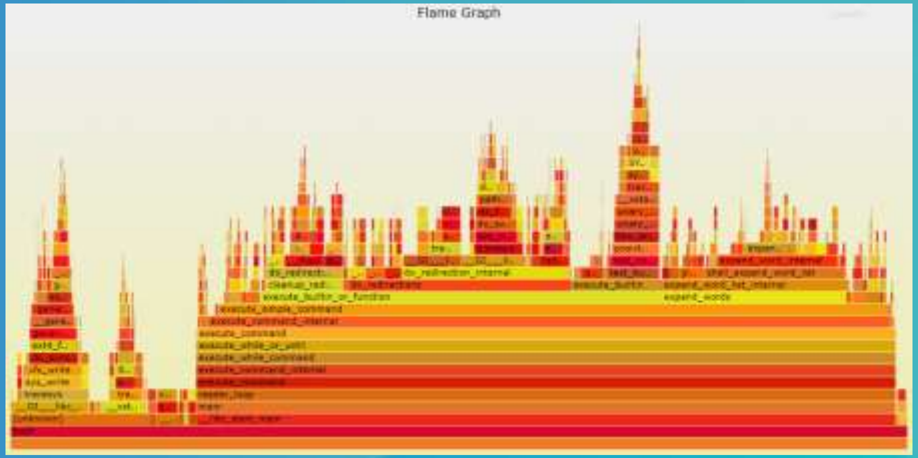
```
cat /proc/meminfo
MemTotal:        15617472 kB
MemFree:           100888 kB
MemAvailable:     6618588 kB
Buffers:             2260 kB
Cached:           5596112 kB
SwapCached:          8276 kB
Active:           6544444 kB
Inactive:         2820204 kB
Active(anon):     3479400 kB
Inactive(anon):    447904 kB
Active(file):     3065044 kB
Inactive(file):   2372300 kB
Unevictable:       126528 kB
Mlocked:           125236 kB
SwapTotal:       16777212 kB
SwapFree:        14524156 kB
Dirty:             134052 kB
Writeback:            152 kB
```

2000：内核统计当前内存使用情况

```
bpftrace -e "tracepoint:kmem:kmalloc {@[kstack(6)]=count();}"

...
@[
    kmalloc_trace+344
    kmalloc_trace+344
    fuse_dentry_init+44
    d_alloc+468
    d_alloc_parallel+68
    __lookup_slow+120
]: 11493
@[
    kmalloc_trace+344
    kmalloc_trace+344
    fuse_alloc_inode+160
    iget5_locked+104
    fuse_iget_backing+124
    fuse_lookup_finalize+88
]: 11514
```

2025：eBPF统计kmalloc调用栈☹



2025：eBPF生成火焰图(2011)，汇总函数执行时间

```
bpftrace -e 'tracepoint:kmem:kmalloc { @=hist(args->bytes_alloc);}'
@:
[128, 256)         78936 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[256, 512)         18962 |@@@@@@@@@@@@                                        |
[512, 1K)          19856 |@@@@@@@@@@@@                                        |
[1K, 2K)            6324 |@@@@                                                |
[2K, 4K)            1682 |@                                                   |
[4K, 8K)            9904 |@@@@@@                                              |
[8K, 16K)            407 |                                                    |
[16K, 32K)           184 |                                                    |
```

2025：eBPF统计kmalloc分配大小/次数☹

eBPF数据的呈现 Counter V.S. Tracing

Counter

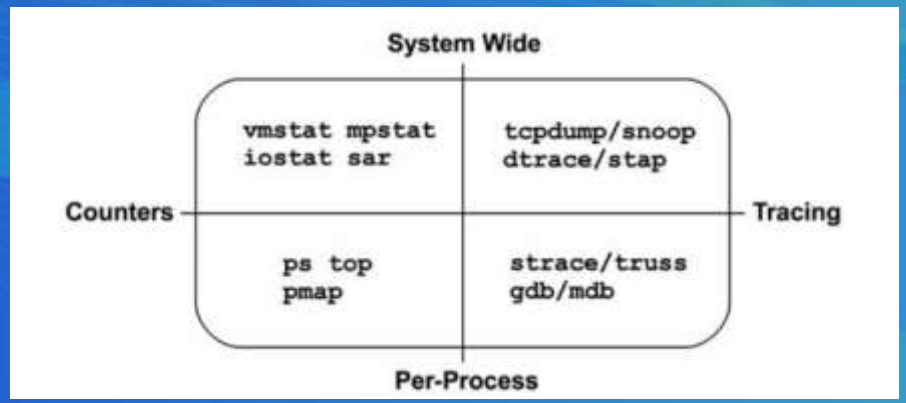
通常为无符号整形，随着事件的发生数量增加，粗粒度但开销低，系统默认开启

Tracing

记录了每次事件的发生，细粒度但开销巨大，系统默认不开启

eBPF具备强大的表达能力，可以实现为上述任意类型，然而真正面对“性能诊断”时

- Counter：数据量太少，无法精确定位
- Tracing：事无巨细，需要花费大量精力寻找
- 最重要的：性能问题成因复杂，最好能够结合上述全部数据，**实现内核和APP的跨层联合分析**



2013:《性能之巅》可观测性工具分类 by Brendan Gregg

Perfetto

Perfetto-谷歌新一代系统性能分析工具

Perfetto

Perfetto 是一个平台级性能跟踪分析与展示工具，提供了

- 记录：系统级和应用级追踪的服务和库，支持本地和 Java 堆分析
- 展示：使用 SQL 分析追踪的库+基于网页的用户界面

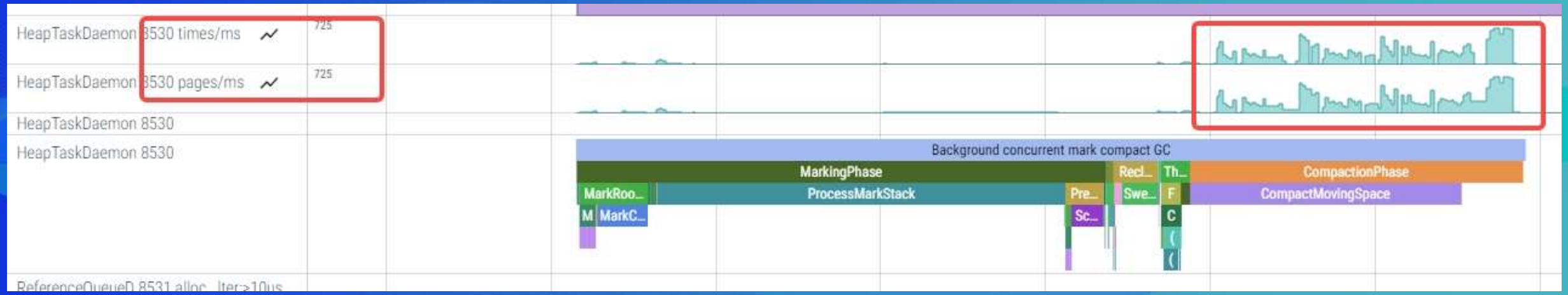
eBPF + perfetto?

perfetto目前不直接支持展示eBPF捕获的事件，因此我们将探索这个问题，进一步释放eBPF潜力

Perfetto-相机启动场景分析

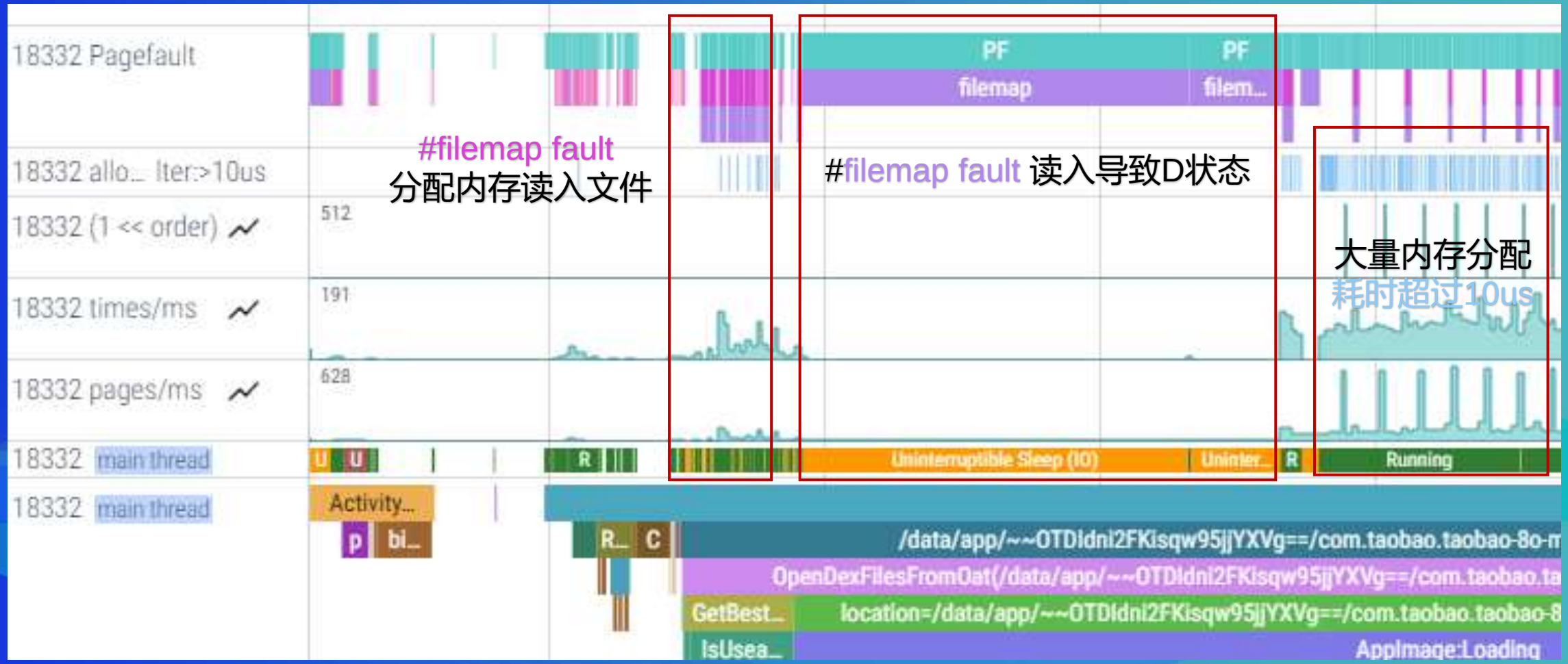
eBPF+Perfetto：GC后段出现大量内存申请

Perfetto 内存分配速度 – GC出现大量内存申请，峰值~725



Perfetto 内存分配速度 – GC内存申请主要集中在GC后段CompactionPhase

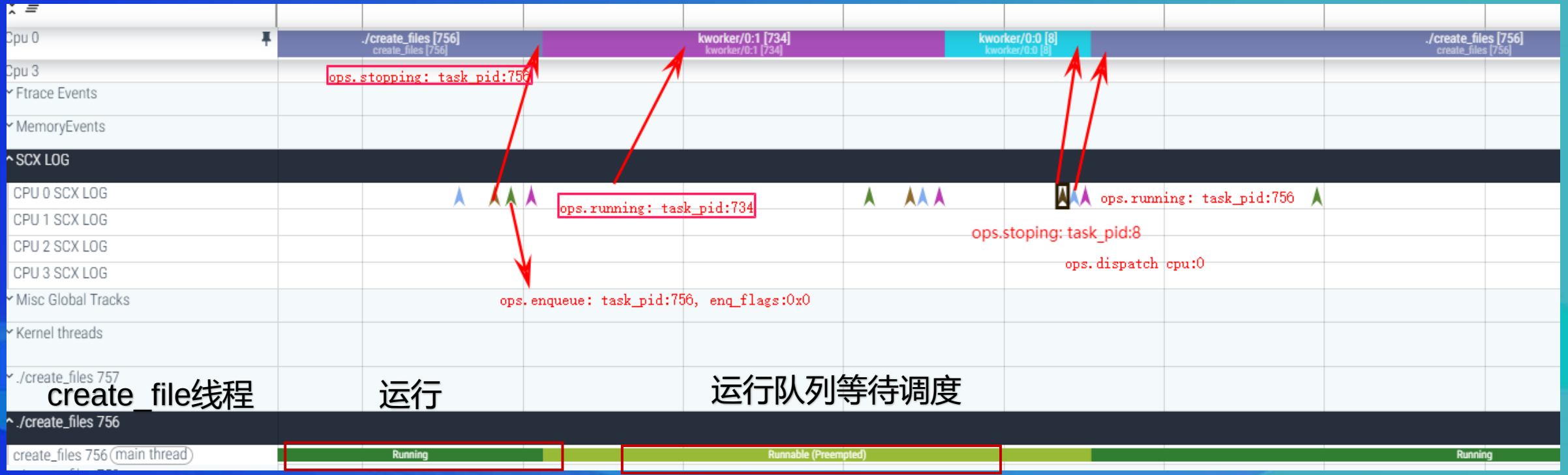
eBPF+Perfetto：缺页异常导致性能波动



Perfetto UI新增#PF / 内存分配tracing可观测性数据

eBPF+Perfetto: sched_ext FIFO调度情况

OSBench create_files[756]程序被kwoker/0:1[734]抢占后，重新加入SCX全局队列等待调度执行



Sched_ext simple FIFO 案例

实现：Perfetto整体架构

Producer Process：高效低开销获取追踪数据，支持多种数据源

Tracing Service：作为核心服务，负责管理追踪会话，协调数据的收集和存储

Buffer：用于临时存储追踪数据，确保数据在处理过程中的高效性和可靠性

Protobuf：作为数据序列化格式，确保数据高效传输和存储

Trace File：追踪数据以文件形式存储，便于后续分析和共享

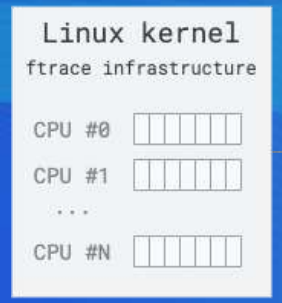
SQL：支持数据快速查询

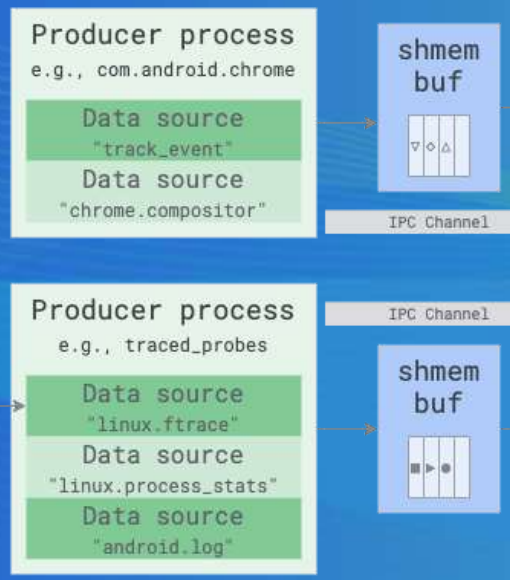
UI：提供可视化界面，用户可以通过它查看和分析追踪数据，帮助识别性能瓶颈

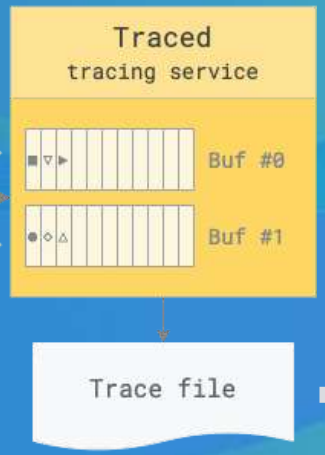
```
data_sources: {
    config {
        name: "linux.ftrace"
        ftrace_config {
                ftrace_events:
                "bpf_trace/*"
                ftrace_events: "ftrace/print"
                disable_generic_events:
                false
                ...
```

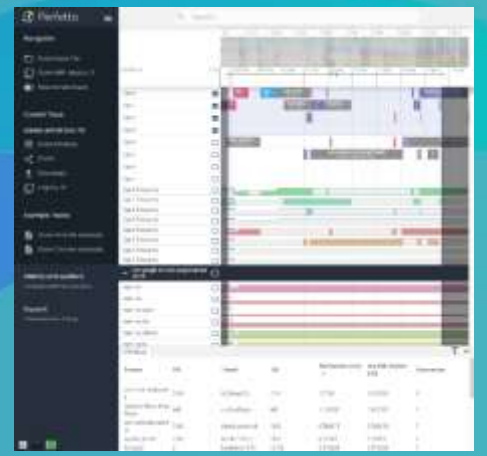
Perfetto开启eBPF相关抓取配置

将eBPF事件写入tracefs









Perfetto将tracefs作为数据源进行抓取

Trace信息汇总为trace文件

Trace文件在UI中展示

实现：eBPF程序获取系统内存分配

eBPF将内核观测数据写入perfetto trace文件（**同步**）

主要挑战：容量规划，合理估计系统压力，控制内核事件输出频率

内核事件输出量过大容易导致性能事件失真、tracing数据丢失

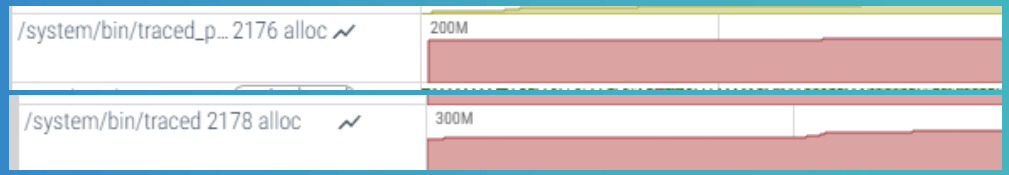
- 设置时间阈值控制关键事件输出
- 设置分配量阈值控制关键事件输出

```c
SEC("tracepoint/kmem/mm_page_alloc")
int handle_mm_page_alloc(void *ctx)
    pfn = ctx->pfn;
    ts = bpf_ktime_get_boot_ns();
    tid = bpf_get_current_pid_tgid();
    pid = tid >> 32;
    // 超过分配时就打印一次
    if (ts - pe->ts >= interval_ms)
        bpf_printk("MAGIC_NUM %16lld\n", pe->nr_pages);

    // 超过分配阈值打印一次
    if (pe->threshold_ctrl > threshold_pgs)
        bpf_printk("MAGIC_NUM %16lld\n", pe->nr_pages);
```

eBPF将内存分配事件写入tracefs

系统压力大，perfetto出现多次未终止binder transaction



Perfetto进程占用内存增加

	pgalloc	kmalloc	sched_switch	syscall
1-2分钟	76	60	23	263
2-3分钟	39	38	19	124
3-4分钟	16	30	12	75
4-5分钟	65	50	20	159
5分钟平均	40	37	15	127

实验室重载 内核事件发生频率 （k times / s）

实现：eBPF程序获取系统内存分配

eBPF将内核观测数据写入perfetto trace文件（**异步**）

使用bpf_ringbuffer/perfbuffer将eBPF获取的内核数据异步提交给用户，用户收到后写入atrace文件

主要挑战：容量规划，合理设计ringbuffer缓冲区大小

```
SEC("tracepoint/vmscan/mm_vmscan_direct_reclaim_end")
int handle_mm_vmscan_direct_reclaim_end(ctx)
{

    event = bpf_ringbuf_reserve(&mm_vmscan_direct_reclaim_end_rb,
                                sizeof(*event), 0);
    if (!event) {
        long sz = bpf_ringbuf_query(&mm_vmscan_direct_reclaim_end_rb,
                                    BPF_RB_AVAIL_DATA);
        bpf_printk("end ringbuf reserve failed, %ld / %ld not consumed\n",
                   sz, 1024*1024);
        return -1;
    }

    ...

    bpf_ringbuf_submit(event, 0);
    return 0;
}
```

eBPF将内存回收事件异步提交给用户

```
Single-producer, consumer/producer competing on the same CPU
==============================================
rb-libbpf          3.045 ± 0.020M/s (drops 3.536 ± 0.148M/s)
rb-custom          3.055 ± 0.022M/s (drops 3.893 ± 0.066M/s)
pb-libbpf          1.393 ± 0.024M/s (drops 0.000 ± 0.000M/s)
pb-custom          1.407 ± 0.016M/s (drops 0.000 ± 0.000M/s)
```

eBPF ringbuf不保证不丢trace

https://patchwork.ozlabs.org/project/netdev/patch/20200529075424.3139988-5-andriin@fb.com/

```
end ringbuf reserve failed, 1048512 / 1048576 not consumed
end ringbuf reserve failed, 1048512 / 1048576 not consumed
end ringbuf reserve failed, 1048512 / 1048576 not consumed
```

eBPF ringbuf 写满，无法reserve内存进行事件提交

```
ATRACE_INT("reclaim", nr_reclaimed);

ATRACE_BEGIN("direct reclaim")
// execute
ATRACE_END()
```

用户使用Android Atrace写入trace文件

实现：Perfetto插件解析

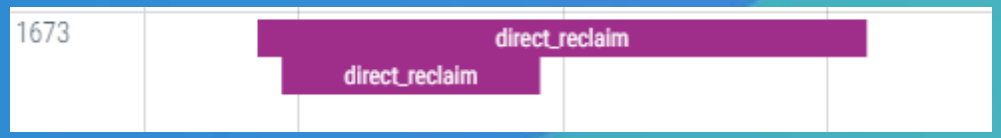
Perfetto UI插件解析eBPF内核关键数据事件，并在perfetto UI中进行呈现

- 输入format字符串 `MAGIC_NUM　　%16lld`，表示内存累积分配页数
- 解析format字符串 `CAST(SUBSTR(display_value, 16, 16) AS INTEGER)`，解析累计分配页数
- 通过ts时间戳对齐android事件

```
DROP TABLE IF EXISTS mm_alloc_table_v2;
CREATE PERFETTO TABLE mm_alloc_table_v2 AS
SELECT
    f.ts,
    t.tid, t.utid, t.name AS tname,
    p.pid, p.upid, p.name AS pname,
    (CAST(SUBSTR(a.display_value,16,16) AS INTEGER) << 12) AS value
FROM ftrace_event f
JOIN args a USING(arg_set_id)
JOIN thread t USING(utid)
JOIN process p USING(upid)
WHERE f.name = 'bpf_trace_printk' AND a.display_value like 'MAGIC_NUM%';
```

Perfetto UI插件中解析内存分配事件

Counter track: 记录进程的累计内存分配-释放

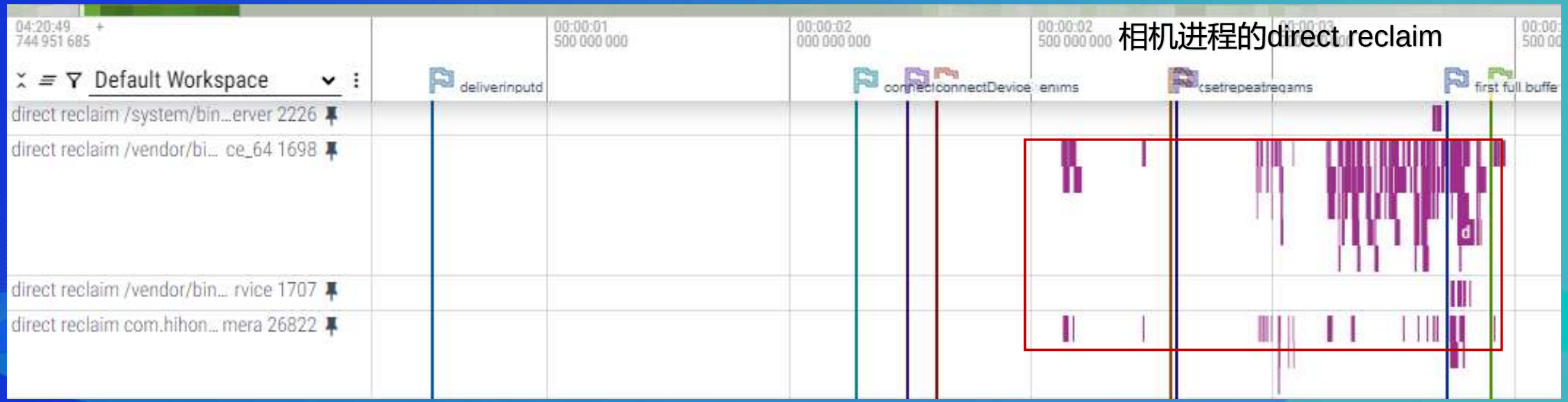


Slice track: 记录进程对应的内核事件执行耗时

效果展示：相机启动性能恶化x%

效果展示：相机启动性能恶化x%

发现后台活跃申请内存xMB，造成相机关键路径出现大量direct reclaim识别相机启动阶段内存异常申请

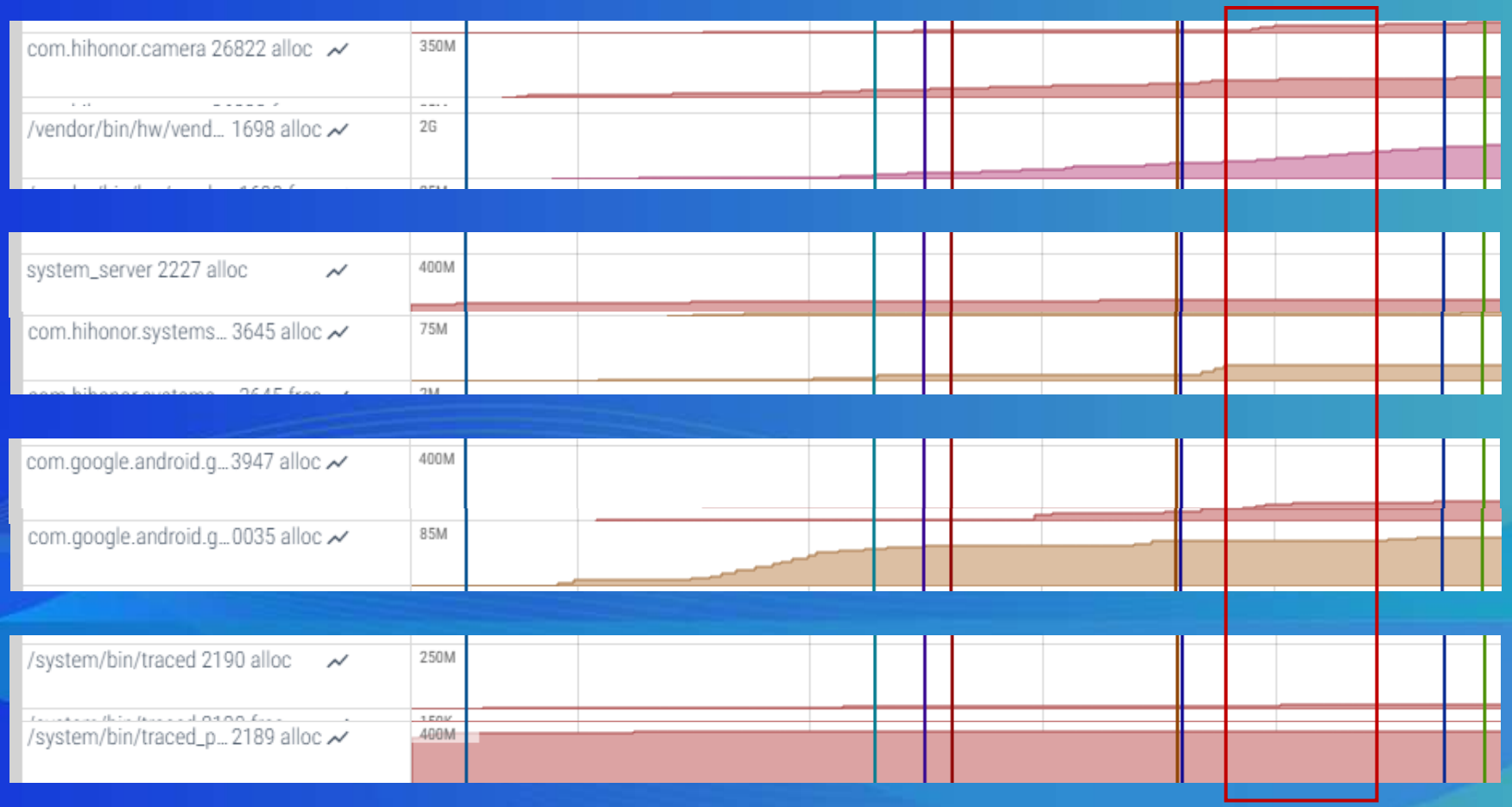


相机点击到预览-配流后出现大量direct reclaim

效果展示：相机启动性能恶化x%

哪些后台活跃申请

集中出现内存direct reclaim



相机

系统

后台活跃

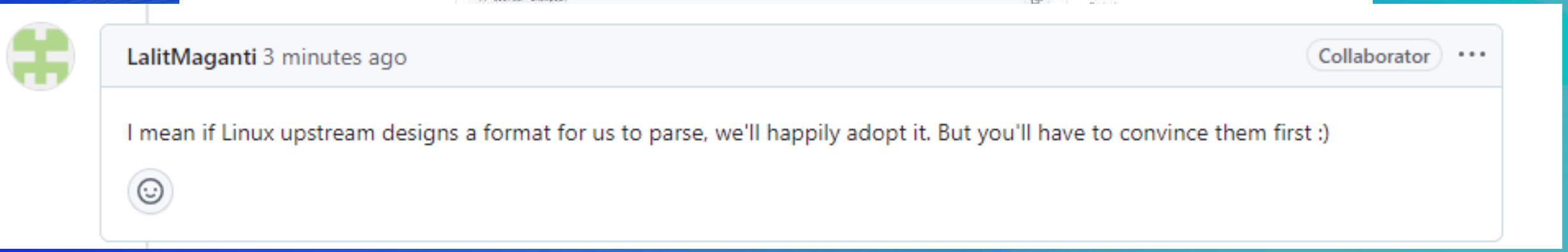
观测工具

识别相机启动阶段后台活跃内存分配

* [Mobicom'24] rethinking process management for interactive mobile systems

未来计划

需要先说服linux upstream约定一个eBPF-perfetto解析协议



Perfetto团队issue-希望加入perfetto eBPF 协议支持

Thank you

HONOR