Integrating and Visualizing eBPF metrics in Performance Co-Pilot & Grafana

by Andreas Gerstmayr

by Neotys

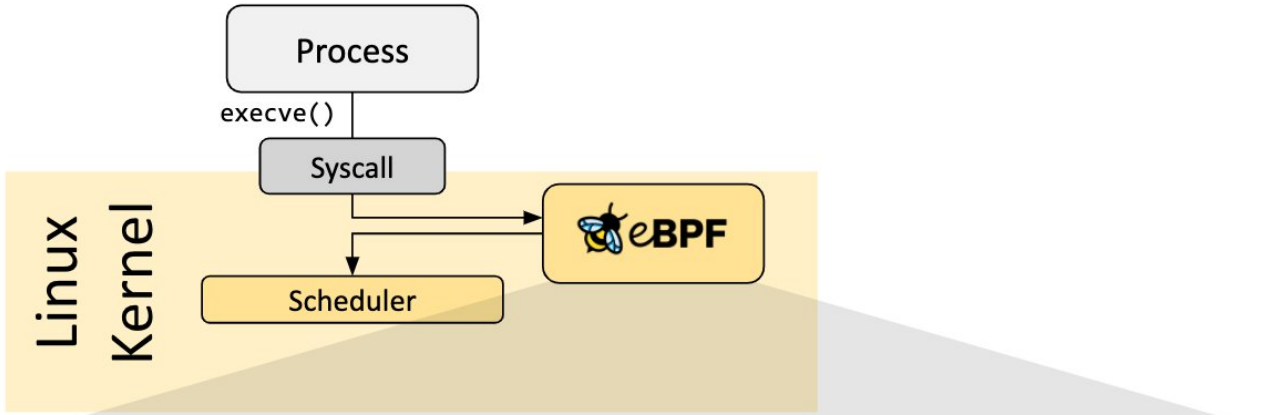
Outline

- What is eBPF?
- Measure block device I/O latency with eBPF
- Integrate the biolatency BCC tool in Performance Co-Pilot
- Visualize block I/O latency with Grafana
- bpftrace introduction
- Outlook

What is eBPF?

- in-kernel virtual machine
- does not require changing kernel sources or loading kernel modules
- sandboxed programs
- intentionally not turing complete, i.e. no (unbounded) loops

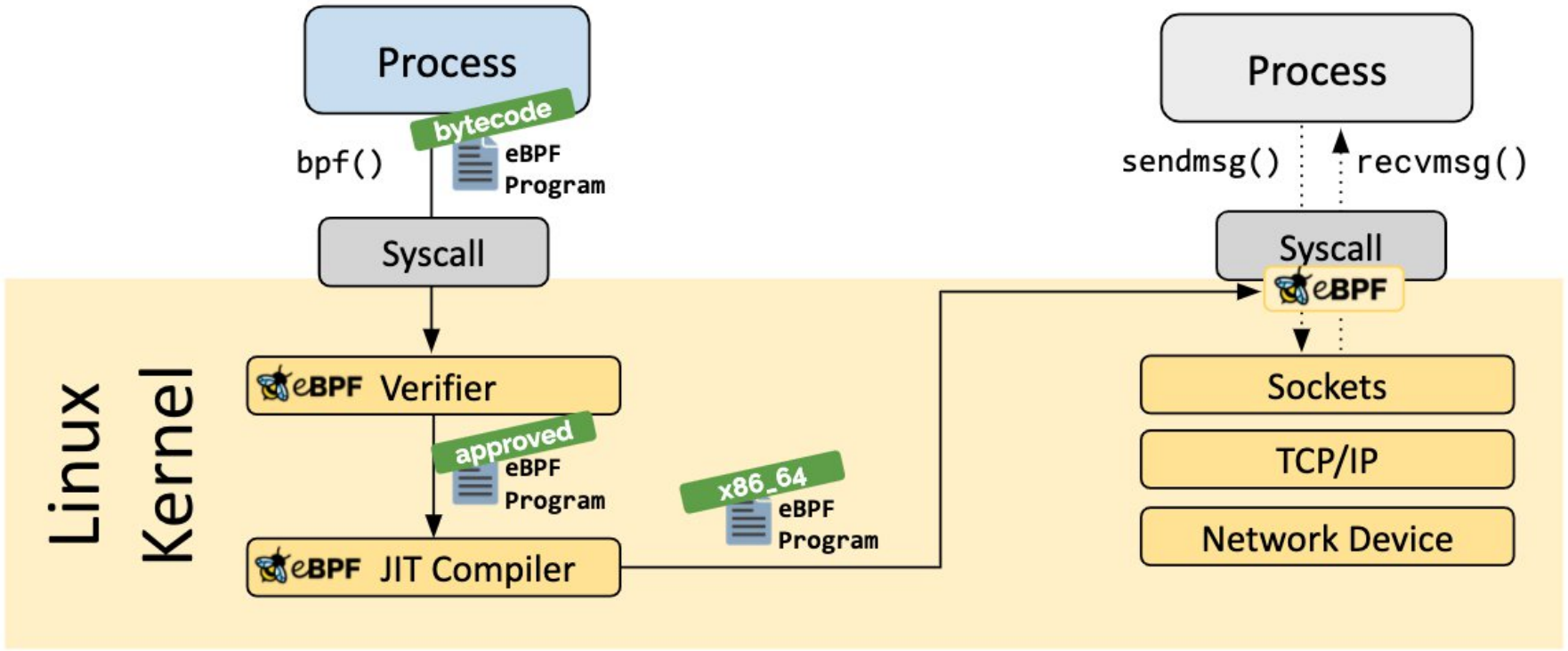
eBPF hooks




```c
int syscall__ret_execve(struct pt_regs *ctx)
{
        struct comm_event event = {
                .pid = bpf_get_current_pid_tgid() >> 32,
                .type = TYPE_RETURN,
        };

        bpf_get_current_comm(&event.comm, sizeof(event.comm));
        comm_events.perf_submit(ctx, &event, sizeof(event));

        return 0;
}
```

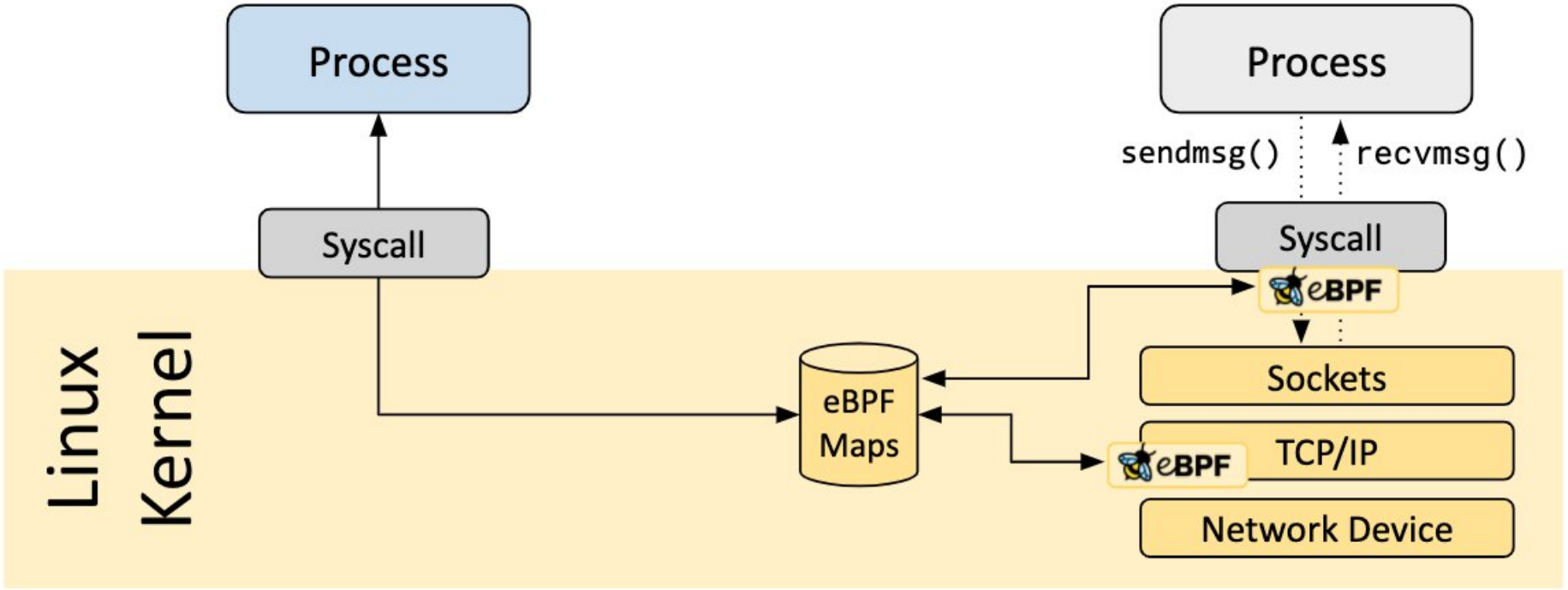
Source: ebpf.io, CC BY 4.0

Loading eBPF programs



Source: ebpf.io, CC BY 4.0

eBPF maps



Source: ebpf.io, CC BY 4.0

BPF Compiler Collection (BCC)

- toolkit for creating eBPF-based tools
- bcc tools are written in Python, embedded eBPF code

BCC tool: biolatency

```
#include <uapi/linux/ptrace.h>
#include <linux/blkdev.h>

BPF_HASH(start, struct request *);
BPF_HISTOGRAM(dist);
```

adapted from the biolatency BCC tool by Brendan Gregg

BCC tool: biolatency

```
int trace_req_start(struct pt_regs *ctx,
                    struct request *req)
{
    u64 ts = bpf_ktime_get_ns();
    start.update(&req, &ts);
    return 0;
}
```

adapted from the biolatency BCC tool by Brendan Gregg

BCC tool: biolatency

```
int trace_req_done(struct pt_regs *ctx, struct request *req) {
    u64 *tsp, delta;
    tsp = start.lookup(&req);
    if (tsp == 0) {
        return 0;   // missed issue
    }
    delta = bpf_ktime_get_ns() - *tsp;
    delta /= 1000;
    dist.increment(bpf_log2l(delta));

    start.delete(&req);
    return 0;
}
```

adapted from the biolatency BCC tool by Brendan Gregg

BCC tool: biolatency

```
b = BPF(text=bpf_text)
b.attach_kprobe(event="blk_mq_start_request",
                fn_name="trace_req_start")
b.attach_kprobe(event="blk_account_io_done",
                fn_name="trace_req_done")

dist = b.get_table("dist")
while True:
    print()
    print(strftime("%H:%M:%S"))
    dist.print_log2_hist("usecs", "disk")
    dist.clear()
    sleep(1)
```

adapted from the biolatency BCC tool by Brendan Gregg

BCC tool: biolatency


```
$ ./biolatency.py

17:53:29
     usecs               : count     distribution
         0 -> 1          : 0        |                                        |
         2 -> 3          : 0        |                                        |
         4 -> 7          : 0        |                                        |
         8 -> 15         : 0        |                                        |
        16 -> 31         : 1        |*                                       |
        32 -> 63         : 2        |***                                     |
        64 -> 127        : 26       |****************************************|
       128 -> 255        : 14       |**********                              |
       256 -> 511        : 0        |                                        |
       512 -> 1023       : 0        |                                        |
      1024 -> 2047       : 1        |*                                       |
      2048 -> 4095       : 0        |                                        |
      4096 -> 8191       : 0        |                                        |
      8192 -> 16383      : 1        |*                                       |
```

BCC tools recap

- great for one-off measurements
- need to SSH into each host and run the test
- no continuous monitoring

Performance Co-Pilot (PCP) to the rescue!

- toolkit for system performance analysis
- 97 included agents (procfs, PostgreSQL, BCC, bpftrace, OpenMetrics, ...)
- metrics have rich metadata (semantics, unit, type, instances)

Integrating the biolatency tool into PCP

- PCP includes a BCC PMDA
- BCC PMDA provides hooks for
  - setup
  - get metric metadata
  - compile eBPF
  - refresh eBPF maps
  - get metric values

BCC PMDA: get metric metadata

```
def metrics(self):
    name = 'disk.all.latency'
    self.items.append(
        # Name - reserved - type - semantics - units - help
        (name, None, PM_TYPE_U64, PM_SEM_COUNTER, units_usecs,
'block io latency distribution'),
    )
    return True, self.items
```

adapted from the biolatency BCC PMDA module by Marko Myllynen

BCC PMDA: compile eBPF

```
def compile(self):
    self.bpf = BPF(src_file=bpf_src)
    self.bpf.attach_kprobe(
        event="blk_mq_start_request",
        fn_name="trace_req_start"
    )
    self.bpf.attach_kprobe(
        event="blk_account_io_done",
        fn_name="trace_req_done"
    )
```

adapted from the biolatency BCC PMDA module by Marko Myllynen

BCC PMDA: refresh eBPF maps

```
def refresh(self):
    dist = self.bpf["dist"]
    self.insts = self.read_log2_histogram(dist,
self.cache)
    dist.clear()
    return self.insts
```

adapted from the biolatency BCC PMDA module by Marko Myllynen

BCC PMDA: get metric values

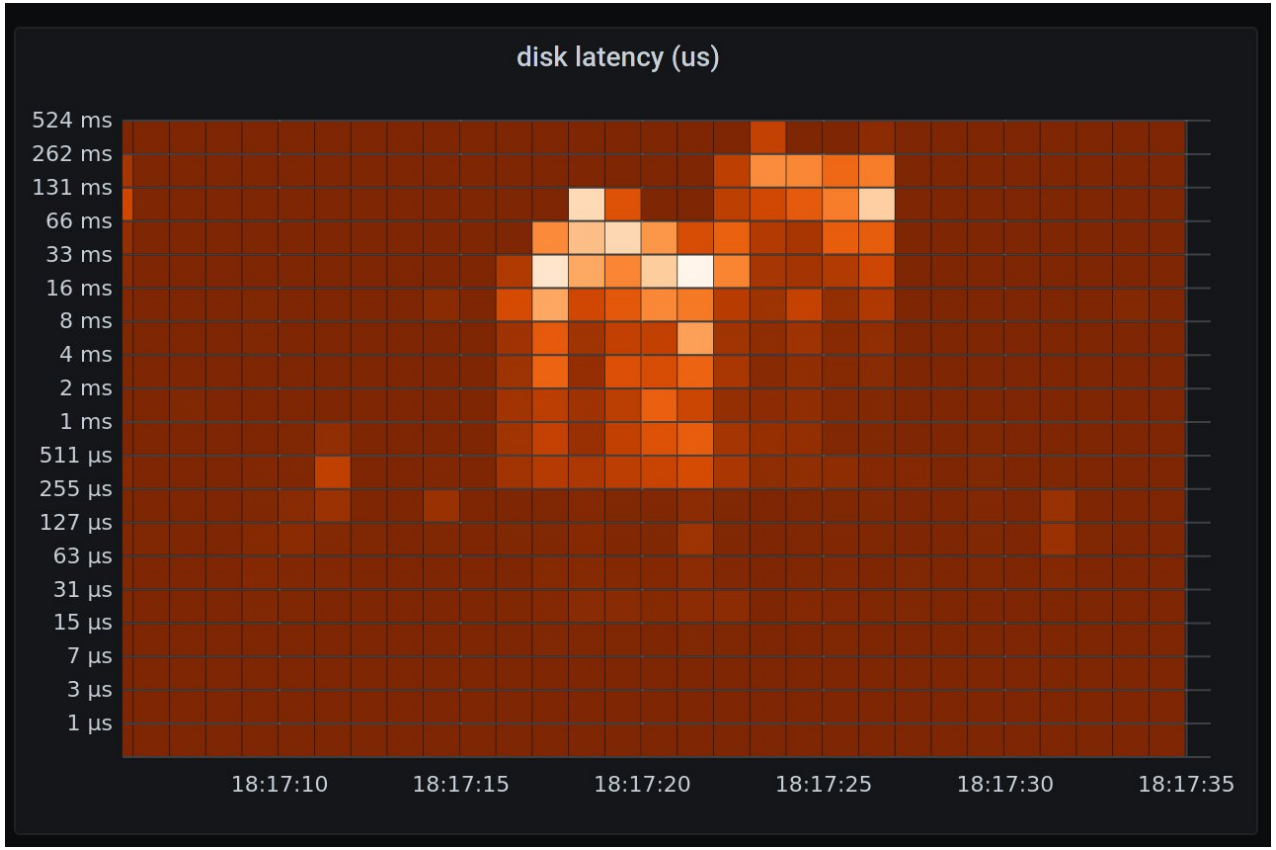
```
def bpfdata(self, item, inst):
    try:
        key = self.pmdaIndom.inst_name_lookup(inst)
        return [self.cache[key], 1]
    except Exception:
        return [PMDA_FETCH_NOVALUES, 0]
```

adapted from the biolatency BCC PMDA module by Marko Myllynen

Display Metric Values (PCP CLI)


```
$ pminfo -f bcc.disk.all.latency

bcc.disk.all.latency
    inst [0 or "0-1"] value 0
    inst [1 or "2-3"] value 0
    inst [2 or "4-7"] value 0
    inst [3 or "8-15"] value 106
    inst [4 or "16-31"] value 1496
    inst [5 or "32-63"] value 3015
    inst [6 or "64-127"] value 6162
    inst [7 or "128-255"] value 5771
    inst [8 or "256-511"] value 9940
    inst [9 or "512-1023"] value 4664
```

Display Metric Values (Grafana)




PCP Vector: eBPF/BCC Overview
now-2m to now+2s
1s
run queue latency (us)
66 ms
33 ms
16 ms
8 ms
4 ms
2 ms
1 ms
511 µs
255 µs
127 µs
63 µs
31 µs
15 µs
7 µs
3 µs
1 µs
19:37:20
19:37:25
19:37:30
19:37:35
19:37:40
19:37:45
19:37:50
19:37:55
19:38:00
19:38:05
19:38:10
19:38:15
19:38:20
19:38:25
19:38:30
19:38:35
19:38:40
19:38:45
19:38:50
19:38:55
19:39:00
19:39:05
19:39:10
19:39:15
19:39:20
disk latency (us)
66 ms
33 ms
16 ms
8 ms
4 ms
2 ms
1 ms
511 µs
255 µs
127 µs
63 µs
31 µs
15 µs
7 µs
3 µs
1 µs
19:37:20
19:37:25
19:37:30
19:37:35
19:37:40
19:37:45
19:37:50
19:37:55
19:38:00
19:38:05
19:38:10
19:38:15
19:38:20
19:38:25
19:38:30
19:38:35
19:38:40
19:38:45
19:38:50
19:38:55
19:39:00
19:39:05
19:39:10
19:39:15
19:39:20

- high-level tracing language for eBPF
- inspired by awk, C, DTrace and SystemTap

biolatency tool in bpftrace

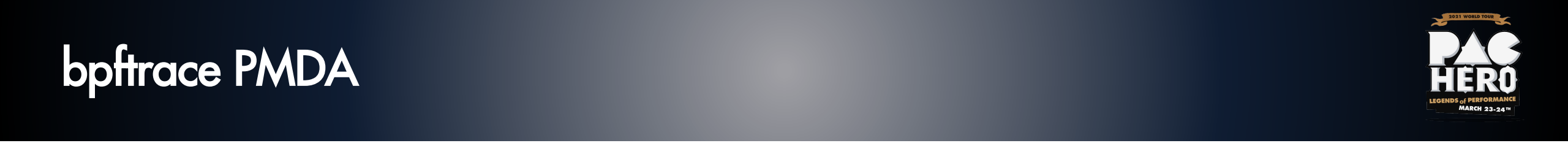
```
// include: @usecs

kprobe:blk_account_io_start
{
        @start[arg0] = nsecs;
}

kprobe:blk_account_io_done
/@start[arg0]/
{
        @usecs = hist((nsecs - @start[arg0]) / 1000);
        delete(@start[arg0]);
}
```

adapted from the biolatency bpftrace tool by Brendan Gregg

bpftrace PMDA

- loads bpftrace scripts from a folder, parses them for metrics and exports them to PCP

bpftrace PMDA




```
$ pminfo bpftrace
bpftrace.scripts.biolatency.data.usecs
bpftrace.scripts.biolatency.data_bytes
bpftrace.scripts.biolatency.code
bpftrace.scripts.biolatency.probes
bpftrace.scripts.biolatency.error
bpftrace.scripts.biolatency.exit_code
bpftrace.scripts.biolatency.pid
bpftrace.scripts.biolatency.status
...
```

bpftrace PMDA


```
$ pminfo -f bpftrace.scripts.biolatency.data.usecs

bpftrace.scripts.biolatency.data.usecs
    inst [7 or "8-15"] value 3
    inst [0 or "16-31"] value 42
    inst [1 or "32-63"] value 92
    inst [2 or "64-127"] value 236
    inst [3 or "128-255"] value 125
    inst [4 or "256-511"] value 252
    inst [5 or "512-1023"] value 39
    inst [6 or "1024-2047"] value 70
    inst [8 or "2048-4095"] value 34
    inst [9 or "4096-8191"] value 33
    inst [10 or "8192-16383"] value 318
    inst [11 or "16384-32767"] value 23
    inst [12 or "32768-65535"] value 16
```

Outlook

- BCC and bpftrace require Clang/LLVM on target host
- lightweight alternative: libbpf & BPF CO-RE

Q & A

Thanks!

github.com/andreasgerstmayr
linkedin.com/in/andreasgerstmayr