第二届中国eBPF开发者大会
WWW.ebpftravel.com

# 基于eBPF的进程生命周期画像工具

工具开发者&汇报人：张子恒(西安邮电大学)

中 国 · 西 安

# 个人介绍

张子恒，西安邮电大学陈莉君教授研二的学生，研究方向Linux内核，内核之旅社区成员，曾获得2023年全国大学生计算机系统能力大赛全国三等奖，2023年开源之夏结项证书

- GitHub主页：
  https://github.com/zhangzihengya
- CSDN主页：
  https://blog.csdn.net/qq_58538265?spm=1018.2226.3001.5343
- eBPF_proc_image项目地址：
  https://github.com/zhangzihengya/lmp/tree/develop/eBPF_Supermarket/CPU_Subsystem/eBPF_proc_image

张子恒

扫一扫上面的二维码图案，加我为朋友。

1、 工具介绍

2、 工具设计与实现

3、 性能测试

4、 测试案例

5、 未来展望

# ① 工具背景

**传统的进程监测工具：**

| 工具 | 简介 | 缺陷 |
| --- | --- | --- |
| top | 数据来源于proc文件系统 | 毫秒级的时间粒度<br>可扩展性差<br>无法提供进程行为数据<br>高负载环境下常常失效 |
| ps | | |
| ..... | | |
| strace | 利用ptrace系统调用来监视并记录正在运行的进程的系统调用 | 功能非常局限<br>文本格式的输出导致处理和分析复杂 |

**eBPF_proc_image 工具旨在清晰地展示出一个进程从创建到终止的完整生命周期，并加入更多的可视化元素和交互方式，使得整个进程画像更加直观、易于理解**

# ② 工具简介

eBPF_proc_image 工具是一款基于 eBPF 技术开发的进程生命周期监测工具，可以展示出的进程生命周期信息类别包括关键时间点信息、持有锁信息、资源使用信息、调度信息、系统调用信息等，该工具具有以下特点：

1. 监控对象可以是单个线程 或 线程组 或 系统中的全部线程
2. 基于函数级的细粒度数据采集
3. 多类别多元素的进程数据展示
4. 结合系统相关信息作为参考和对比
5. 支持预先按需挂载，使用时激活，可动态调整数据采集策略
6. 基于 Prometheus 和 Grafana 的可视化平台，数据存储支持动态可扩展的表头

# ② 工具简介

## proc_image：挂载函数，等待输出

```
Usage: proc_image [OPTION...]
Trace process to get process image.

  -a, --all                  Attach all eBPF functions(but do not start)
  -k, --keytime              Attach eBPF functions about keytime(but do not
                             start)
  -l, --lock                 Attach eBPF functions about lock(but do not start)

  -r, --resource             Attach eBPF functions about resource usage(but do
                             not start)
  -s, --syscall              Attach eBPF functions about syscall sequence(but
                             do not start)
  -S, --schedule             Attach eBPF functions about schedule (but do not
                             start)
  -?, --help                 Give this help list
      --usage                Give a short usage message
```

## controller：控制数据采集策略

```
Usage: controller [OPTION...]
Trace process to get process image.

  -a, --activate             Set startup policy of proc_image tool
  -c, --cpuid=CPUID          Set For Tracing  per-CPU Process(other processes
                             don't need to set this parameter)
  -d, --deactivate           Initialize to the original deactivated state
  -f, --finish               Finish to run eBPF tool
  -k, --keytime=KEYTIME      Collects keytime information about
                             processes(0:except CPU kt_info,1:all kt_info,any 0
                             or 1 when deactivated)
  -l, --lock                 Collects lock information about processes
  -m, --myproc               Trace the process of the tool itself (not tracked
                             by default)
  -p, --pid=PID              Process ID to trace
  -P, --tgid=TGID            Thread group to trace
  -r, --resource             Collects resource usage information about
                             processes
  -s, --syscall=SYSCALLS     Collects syscall sequence (1~50) information about
                             processes(any 1~50 when deactivated)
  -S, --schedule             Collects schedule information about processes
                             (trace tool process)
  -t, --time=TIME-SEC        Max Running Time(0 for infinite)
  -?, --help                 Give this help list
      --usage                Give a short usage message

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.
```

1、 工具介绍

2、 工具设计与实现

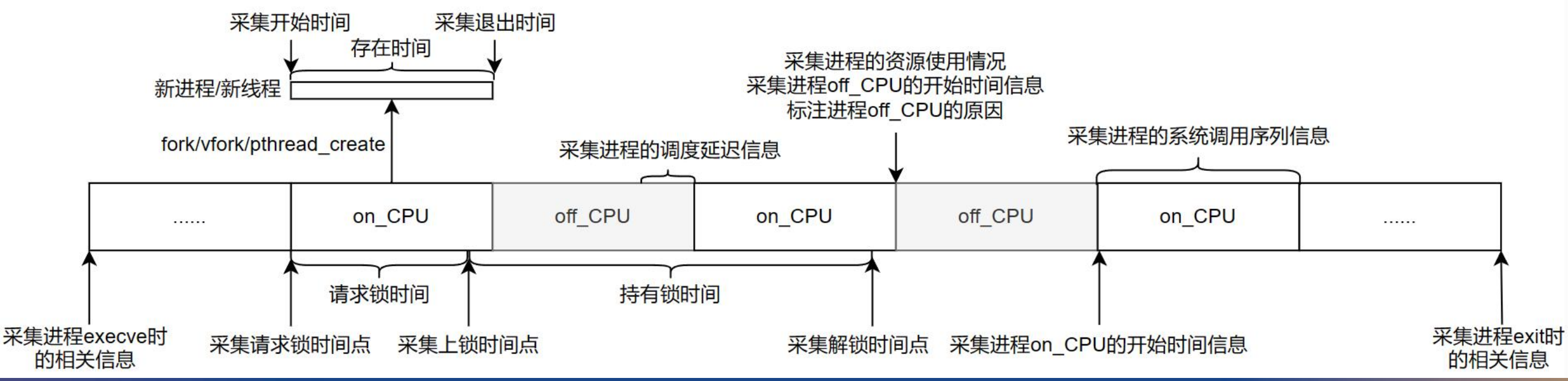
3、 性能测试

4、 成功展示

5、 未来展望

# ① 采集指标

## 进程数据采集时序图：




## 进程的数据可归为5种类型：

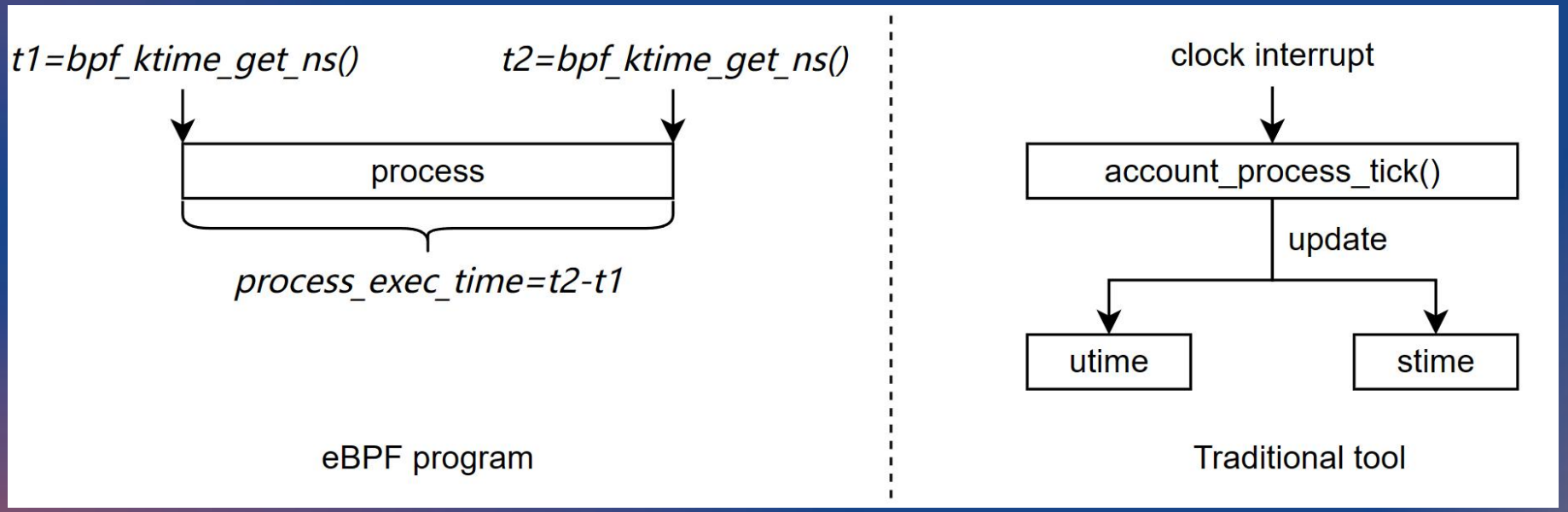
**关键时间点信息、持有锁信息、资源使用信息、调度信息、系统调用信息**

# ② 细粒度的时间信息采集

时间信息是进程生命周期中最重要的信息

时间信息的采集统一采用 bpf_ktime_get_ns()，以实现纳秒级的粒度

以采集进程的执行时间为例，相比与基于proc文件系统的传统工具，如下图所示：



# ③ 技术选择

## 针对进程生命周期数据的采集：

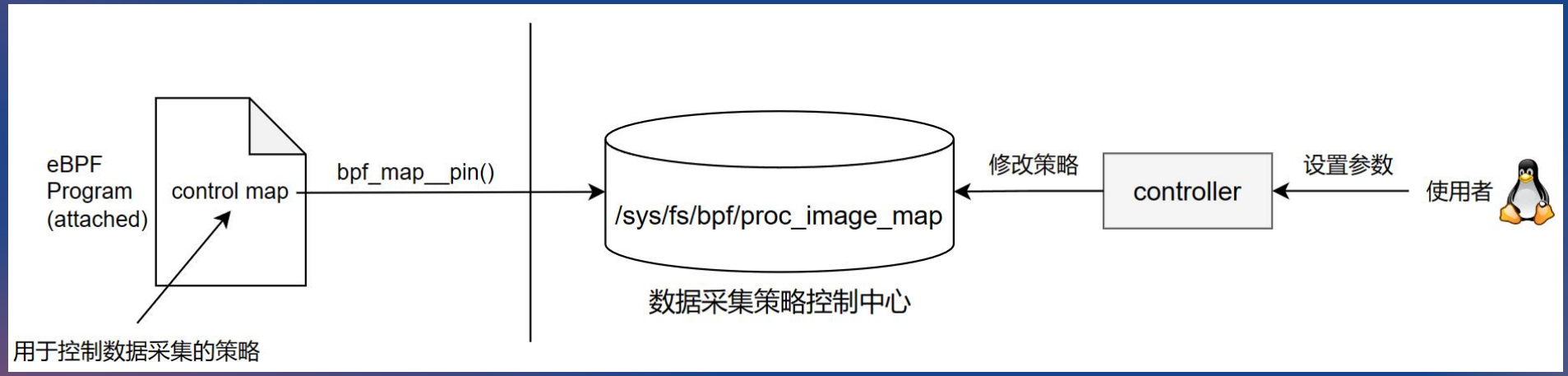
| | 内核模块 | libbpf |
|---|---|---|
| 在功能实现方面 | 可以直接访问内核资源，以实现复杂的功能 √ | 虽然存在一些限制，但足以满足进程数据采集的要求 √ |
| 在安全性方面 | 内核提供的保护机制和错误处理机制不足以发现和应对所有的bug | 严格的代码逻辑验证与内核其他组件隔离 √ |
| 在可移植性方面 | 不同的内核版本适配工作很繁琐 | BPF CO-RE(Compile Once – Run Everywhere) √ |
| 在工具编程方面 | 繁琐复杂 | 框架简单易上手 √ |

# ④ 预先挂载使用激活

一般的eBPF程序执行流程：load->attach->collect->output

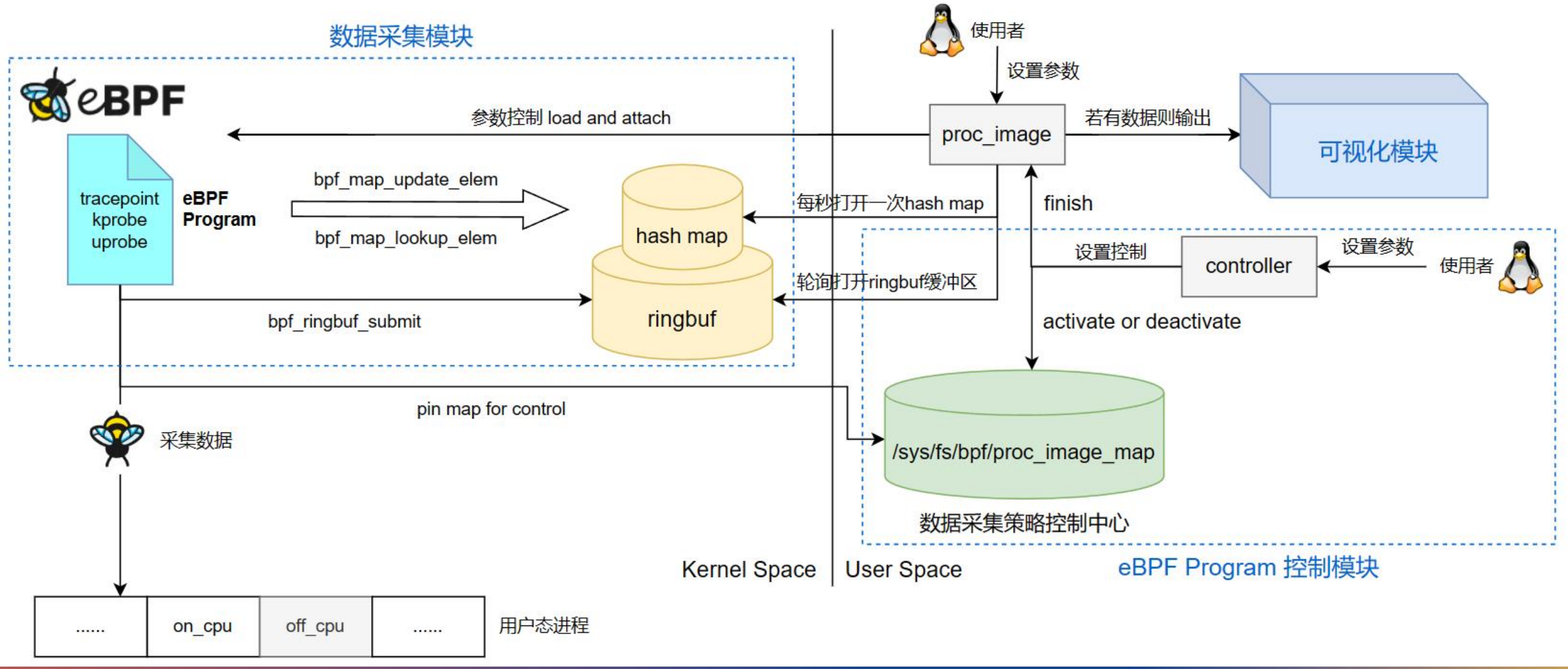
- 一键式运行，可能导致在高负载场景下存在较大运行延迟
- 不支持动态修改数据采集的策略
- 一旦运行数据类型及格式输出固定，无法多元素地进行可视化

为了解决这些问题，eBPF_proc_image 工具结合 pin map：

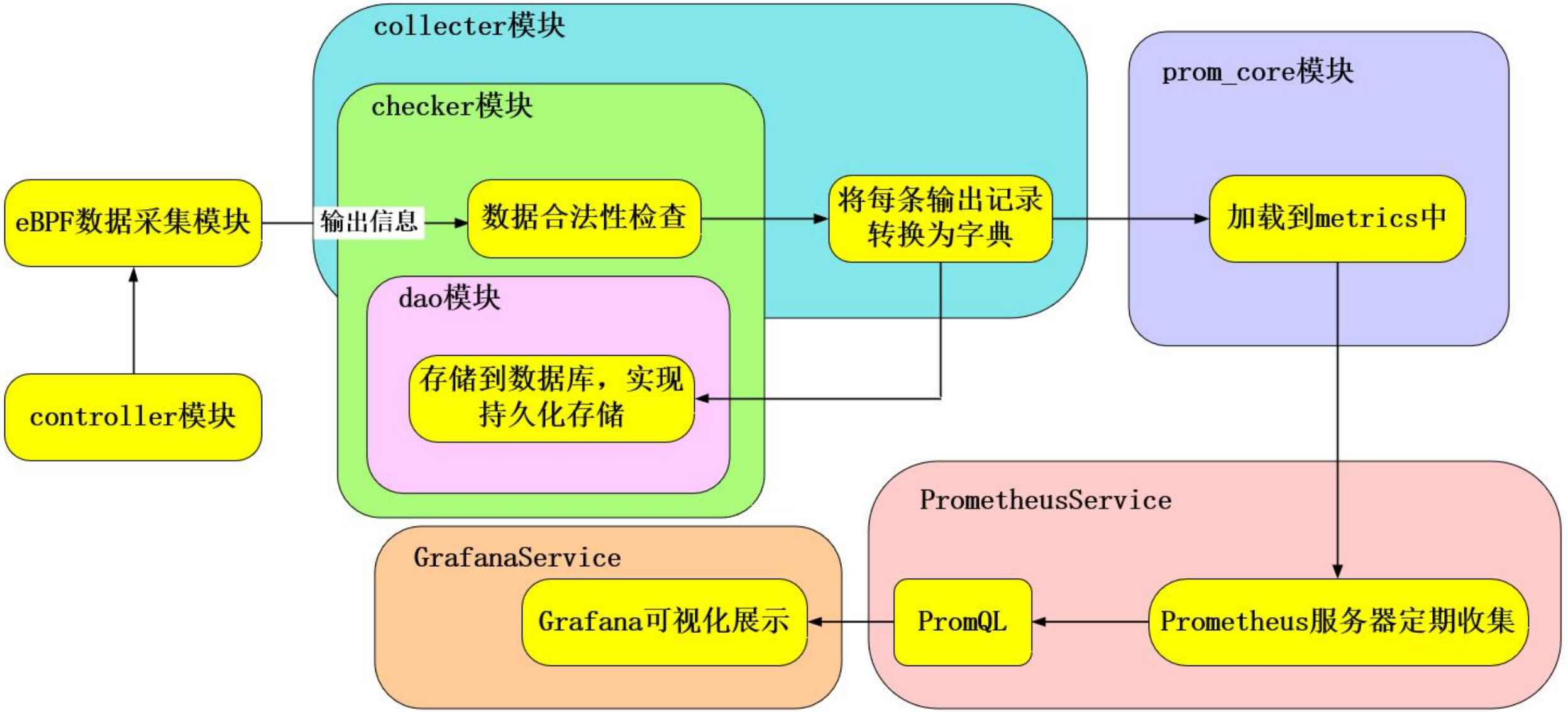



- 支持预先按需挂载，以满足在高负载场景下直接使用
- 使用时激活，可动态调整数据采集策略，满足了多元素可视化的需求

# ⑤ 工具框架图

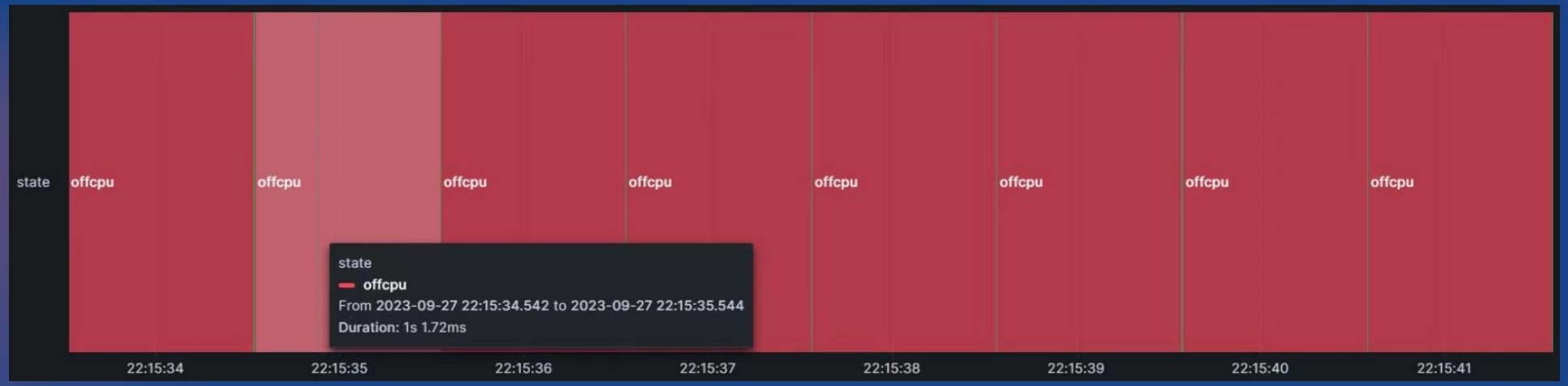

数据采集模块
eBPF
tracepoint
kprobe
uprobe
eBPF Program
参数控制 load and attach
bpf_map_update_elem
bpf_map_lookup_elem
hash map
ringbuf
bpf_ringbuf_submit
pin map for control
采集数据
使用者
设置参数
proc_image
若有数据则输出
可视化模块
每秒打开一次hash map
finish
轮询打开ringbuf缓冲区
设置控制
controller
设置参数
使用者
activate or deactivate
/sys/fs/bpf/proc_image_map
数据采集策略控制中心
Kernel Space
User Space
eBPF Program 控制模块
.....
on_cpu
off_cpu
.....
用户态进程

# ⑥ 基于 Prometheus 和 Grafana 的可视化平台



# ⑦ 数据可视化设计

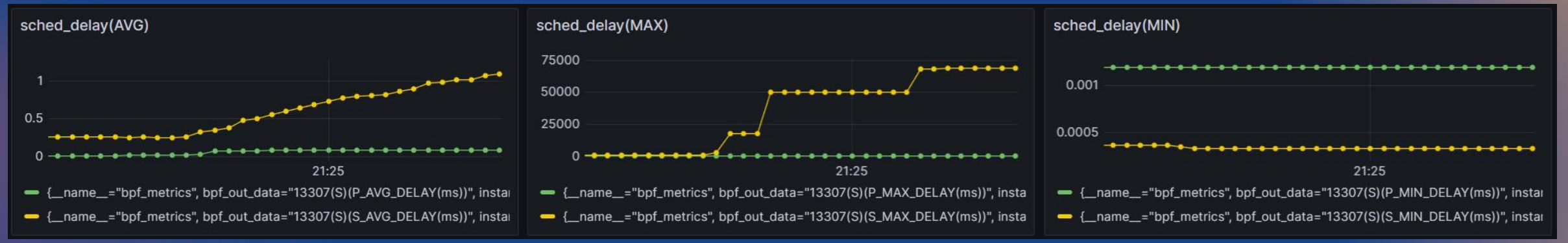
## 进程上下CPU可视化设计：



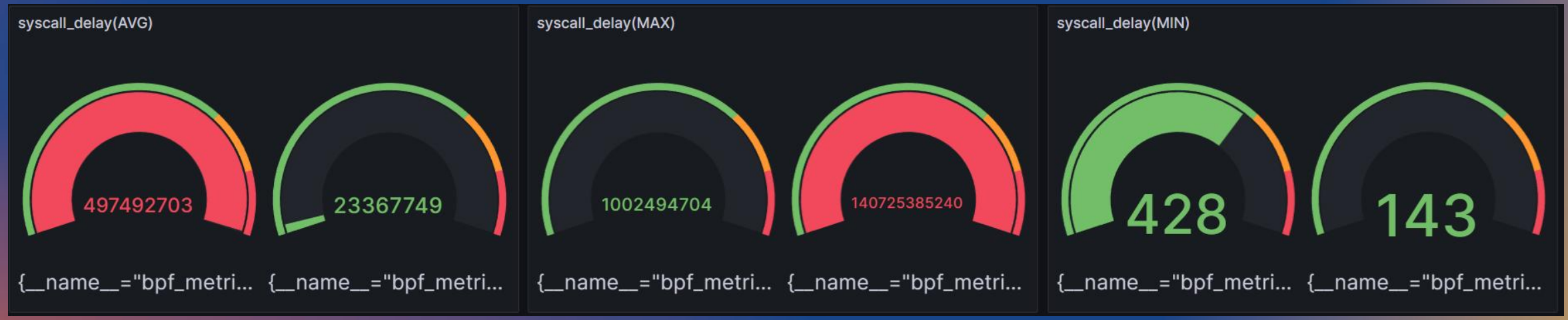

## 进程资源使用情况可视化设计：

# ⑦ 数据可视化设计

## 进程调度信息可视化设计：




## 进程系统调用可视化设计：

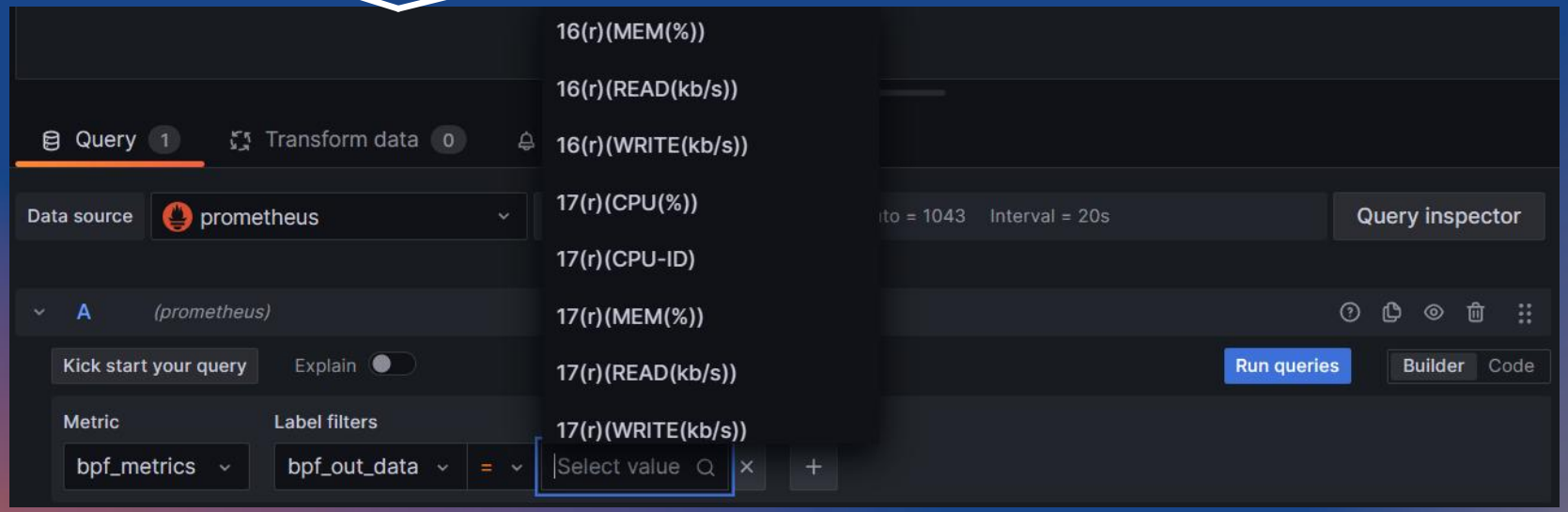


# ⑧ eBPF_proc_image action

```
Successful to compile eBPF_proc_image tools:
proc_image --- Pre-attach to eBPF program, data is collected and output after activation
controller --- Controller for proc_image tool
Please start your use ~
```

step one **data-visual** + **eBPF程序** ← step two **控制器**

step three



1、 工具介绍

2、 工具设计与实现

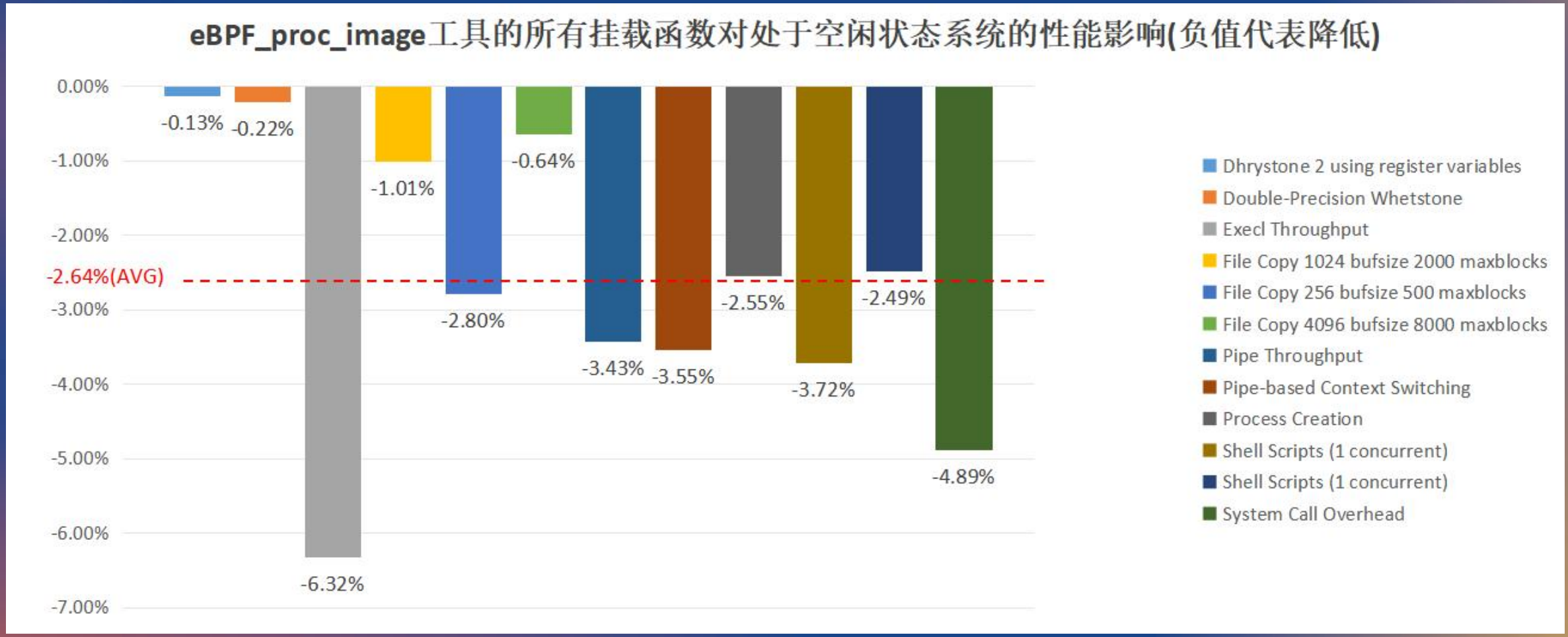
3、 性能测试

4、 成果展示

5、 未来展望

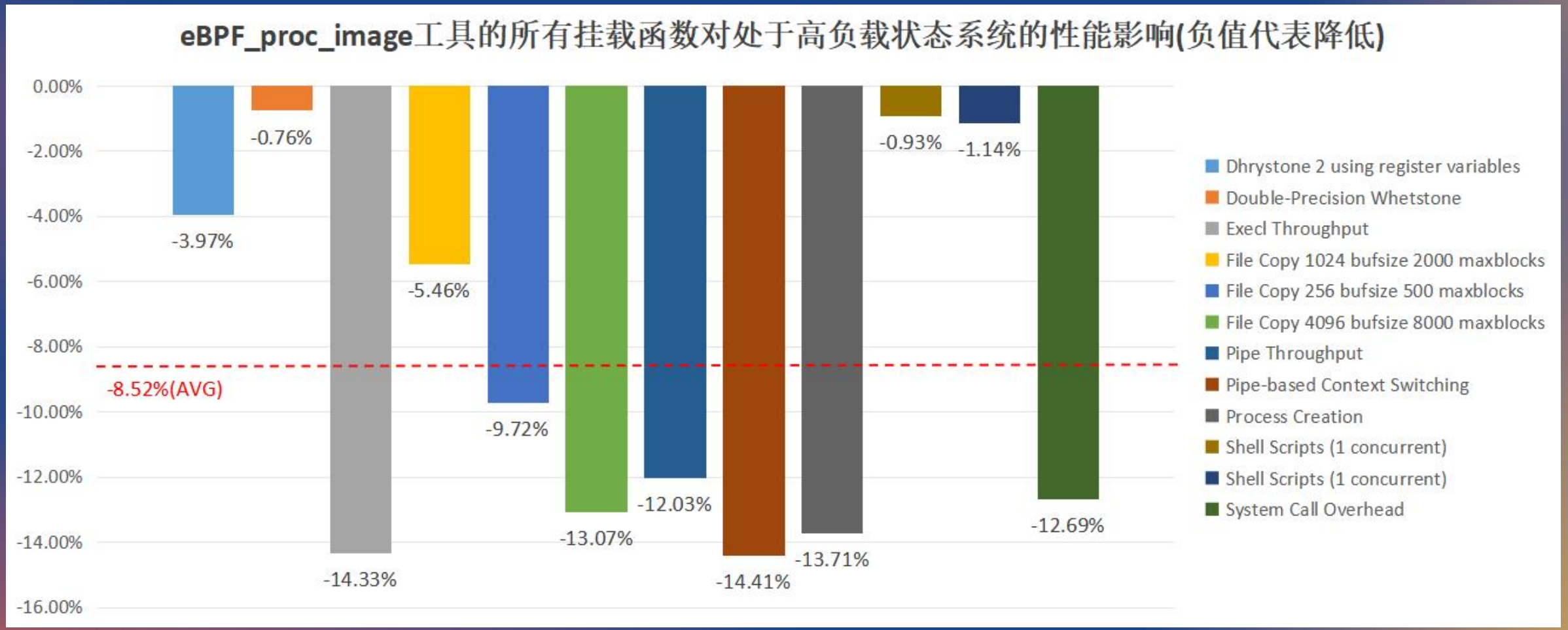
# ① 系统处于空闲状态下的测试

## 性能测试工具：UnixBench



# ② 系统处于高负载状态下的测试

**性能测试工具：UnixBench**　　**压测工具：stress-ng**



1、 工具介绍

2、 工具设计与实现

3、 性能测试

4、 测试案例

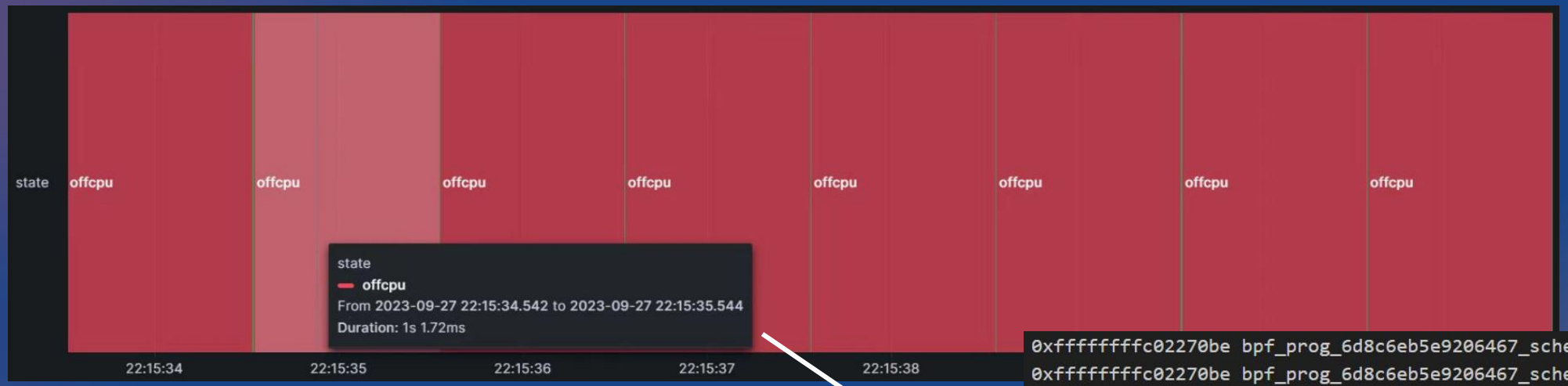
5、 未来展望

# ① 进程阻塞原因的定位

**测试逻辑：**

执行 while 循环实现每上一次 CPU 睡眠一秒，这一秒代表阻塞时间

**可视化结果：**




**offcpu调用栈信息：**

```
0xffffffffc02270be bpf_prog_6d8c6eb5e9206467_sched_switch+0x1de
0xffffffffc02270be bpf_prog_6d8c6eb5e9206467_sched_switch+0x1de
0xffffffffb58cc159 bpf_trace_run4+0x59
0xffffffffb573ec90 __bpf_trace_sched_switch+0x10
0xffffffffb6755cc3 __schedule+0x363
0xffffffffb6756133 schedule+0x63
0xffffffffb675d48d do_nanosleep+0x5d
0xffffffffb57f129f hrtimer_nanosleep+0xbf
0xffffffffb57fb923 common_nsleep+0x43
0xffffffffb57fe8f1 __x64_sys_clock_nanosleep+0xf1
0xffffffffb673ee29 do_syscall_64+0x59
0xffffffffb68000e6 entry_SYSCALL_64_after_hwframe+0x6e
```

# ② 进程调度延迟的实时监控

**测试场景：**

通过 stress-ng 工具启动与主机 CPU 核心数相同数量的 CPU 密集型进程，以使 CPU 被充分利用

**测试对象：**

监测 top 进程的调度延迟信息

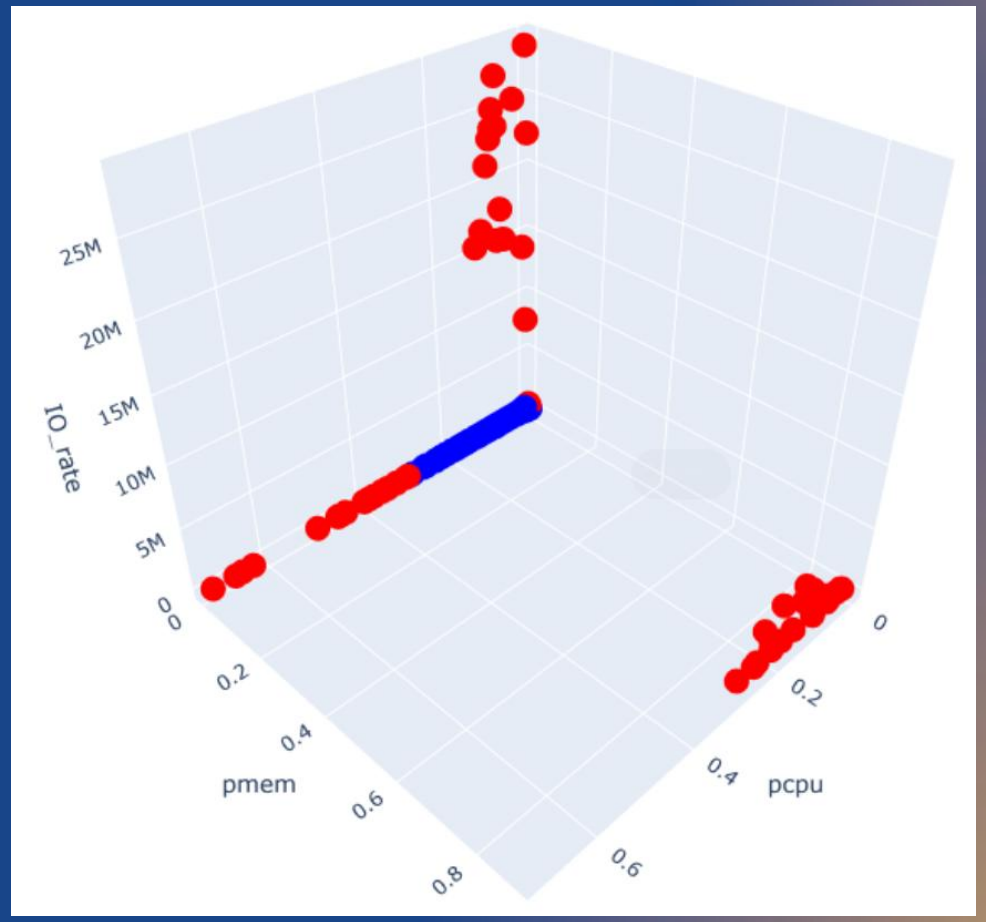
**可视化结果：**

# ③ 进程资源使用情况的异常检测

异常进程的模拟：

- CPU 密集型进程 —— 使用stress-ng工具的-c参数模拟高CPU利用率进程
- mem 密集型进程 —— 使用stress-ng工具的-vm参数模拟高内存利用率进程
- write 密集型进程 —— 使用stress-ng工具的-iomix和-iomix-bytes参数模拟高速写入进程
- read 密集型进程 —— 使用dd工具模拟高速读取进程

实验结果：



1、 工具介绍

2、 工具设计与实现

3、 性能测试

4、 成果展示

5、 未来展望

# 未来展望

- 在真实的应用场景下进行完善和优化
- 进行更加生动的进程画像
- 结合 bpftool 工具，实现更加灵活的 eBPF_proc_image 工具的使用
- 加入异常检测模块，以及时准确的发现异常
- 结合 bpftime 工具，针对进程异常行为设置热补丁

# 感谢各位专家的倾听

# THANKS