CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

BPF用户态回溯lib和持续Profiling系统

字节跳动-章雨宸
zhangyuchen.lcr@bytedance.com

CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

目录

CONTENTS

内核BPF ustack回溯现状

需要开启frame-pointer

内核BPF仅支持使用bpf_get_stack_id，使用frame-pointer进行回溯．但这个frame-pointer开启后有一些性能损失，所以大部分程序都不会开frame-pointer编译．

依赖重编译

内核问题排查场景下，常常需要看用户态应用的调用路径．例如某个内核文件是走什么路径调用到的．需要重编整个调用链的二进制到fp．

为什么没有开源的用DWARF回溯的方案?

内存占用问题

- **debuginfo的数据占用较大**：debuginfo主要是symbols和dwarf，dwarf中除CFI外还包含多种debug信息.
- **多进程回溯不能复用**：BPF更多是多进程上下文，现有的解析和回溯库都不支持debuginfo跨进程复用，内存占用大.

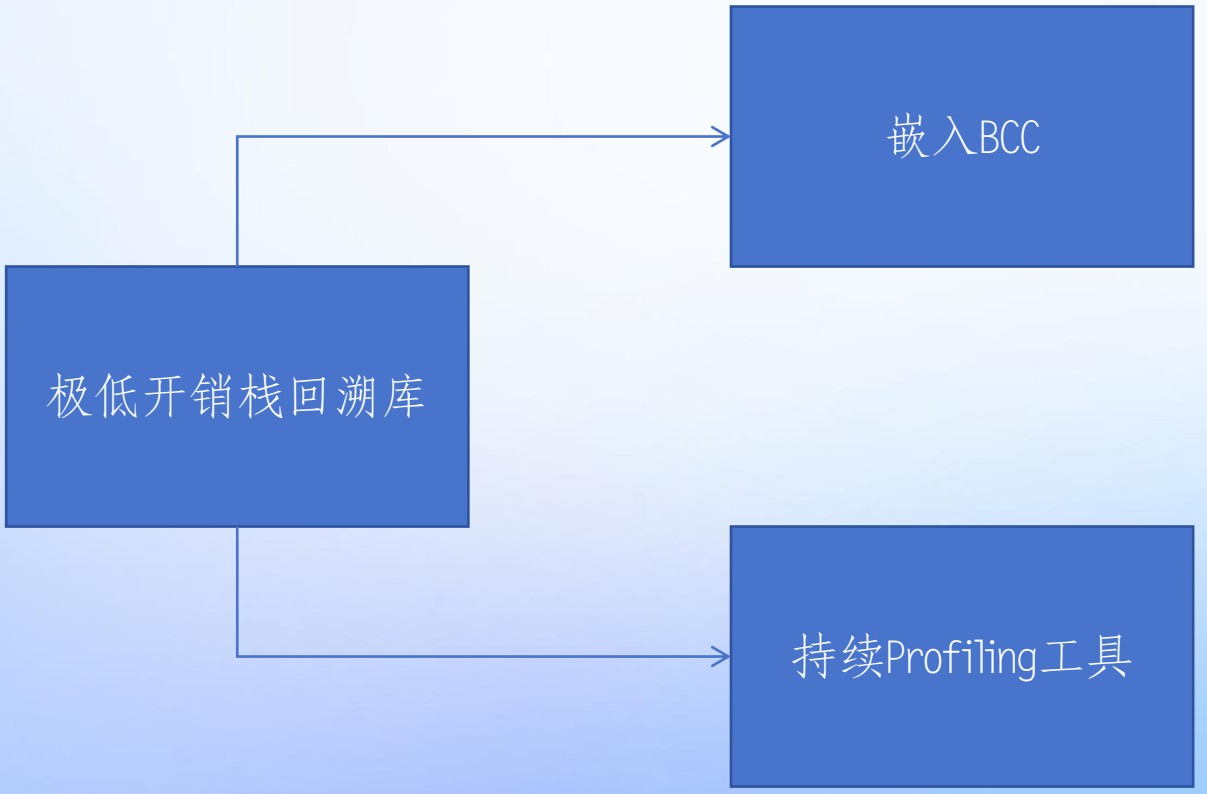
缓存穿透问题

现有回溯库普遍采用缓存模式，缓存穿透会导致磁盘I/O，使解析速率出现不可预测的波动，难以满足生产环境对稳定低损耗的严苛要求。

核心是现有的解析库无法支持整机维度的ustack解析支持.

CFI：核心的回溯堆栈展开信息. 可以展开为表达式来根据pc和sp计算上一层栈帧位置

现有组件



和BCC结合，用于灵活排查涉及用户态栈的问题的场景

持续Profiling无侵入采集，可用于保留历史Profiling数据排查偶发抖动问题

为什么Perf不可以做到持续Profiling?

讨论这个论题是想要突出这个回溯库的性能提升有什么价值.

典型Perf使用逻辑

perf.data size = (8K~64K) * freq * cpu数量 → 19hz 128核 选择最低的8K采集 = 19MB/s = 1680 TB / 天

极低的开销给持续Profiling带来了什么

性能够好后，解析转移到运行时，在线解析，保留文本结果

结合一些保存上的优化，可达到100MB - 200MB / 天的存储开销，且保留Pod / cpu号 / thread comm / cgroup id这些细粒度信息
真正可在生产环境部署. 无侵入，无性能损失，整机维度.

结果

- 极大减轻存储开销，保留的是文本栈信息而不是原始栈内容
- 同时减轻写入开销和取消离线解析步骤，可做到持续解析.

效果展示 – 持续Profiling工具

我们主要以持续Profiling为例，讲述这个lib的应用价值. 同理可推到嵌入bcc的版本.

核心优势 – 极低的开销

典型业务场景	机器规格	活跃CPU使用量 (关联CPU占用)	活跃elf数量 (关联内存占用)	CPU占用	内存占用	存储占用
clickhouse独占机器	128C / 1.5T	67C	253	0.10C	2.5G	100M / 天
mysql独占机器	128C / 1.0T	14C	384	0.02C	1.0G	173M / 天
容器场景(单机100 Pod)	256C / 1.0T	143C	2679	0.26C	6.9G	470M / 天

理论1核可支持400活跃核解析，最极端的极多容器(elf种类多)场景 12G左右内存占用，正常业务2G左右

其他优势

- 理论和大范围部署应用均显示完全不影响业务性能.
- 对于业务进程无侵入，不需要修改. 不需要特意保留debuginfo.
- 可保留采集时间内每秒的火焰图.
- 面向单机维度，无任何外部和网络依赖. 单个二进制文件即可运行.
- 保留Pid / Comm / Thread Comm / cgroup id / pod name / cpu.
- 默认为整机维度所有进程均记录性能表现.


基本信息
IP: 2605:340:cd51:4500:3c63:3942:a296:b475
2025-10-16 14:44:21 ~ 2025-10-16 15:14:21
64 cores | x86_64 | 5.4.210.bsk.6-sign-amd64 | Debian GNU/Linux 10 (buster)
编辑备注
Vela监控
跳转至Vela
时序分析
时间选择区间大小
1s
快速时间区间选择
异常区间选择
直接选择异常/最大值区间
CPU相关目标筛选
添加筛选项
网络相关筛选
添加筛选项
异常提示：存在CPU 利用率突增，跳转至相关时间 | 存在irqoff cpu占用，该指标对时延敏感业务存在较大影响，查看相关堆栈
操作提示：控制键盘「」「」切换至同大小邻近时间区间 | 指标说明
2025-10-16 14:51:22
CPU 利用率 36.51% (23.37 c)
CPU-内核态占比 9.46% (2.21 c)
单击鼠标/触控板查看该时间节点所在区间的统计/堆栈数据
2025-10-16 14:44:21
2025-10-16 14:47:19
2025-10-16 14:50:17
2025-10-16 14:53:15
2025-10-16 14:56:13
2025-10-16 14:59:11
2025-10-16 15:02:09
2025-10-16 15:05:07
2025-10-16 15:08:05
2025-10-16 15:11:03
2025-10-16 15:14:01
统计分析 2025-10-16 14:50:39
堆栈详情分析 2025-10-16 14:50:39
函数搜索
输入栈帧关键词
显示标签
binary comm
火焰图筛选
添加筛选项
数据展示
显示栈顶函数表格
渲染类型
火焰图
函数关系图
火焰图方向
栈底聚合
栈顶聚合
显示配置
堆叠展示聚合标签
只展示内核态函数
忽略栈帧中CPU信息
数据下载
火焰图文本数据
栈顶函数数据
当前火焰图PNG
libunwind::CFI_Parser::parseCIE
libunwind::CFI_Parser::findFDE
libunwind::UnwindCursor::getInfoFromDwarfSection
libunwind::UnwindCursor::setInfoBasedOnIPRegister
libunwind::UnwindCursor::step
unw_backtrace
Poco::Exception::Exception
DB::Exception::Exception
DB::BroadcastSenderProxy
DB::ExchangeUtils::waitAccept
DB::BrpcExchangeReceiverRegistryService::registry
brpc::policy::ProcessRpcRequest
brpc::InputMessenger::OnNewMessages
brpc::Socket::ProcessEvent
brpc::ProcessInputMessage
bthread::TaskGroup::task_runner
bthread_make_fcontext
ThreadPoolImpl::ThreadFromThreadPool::worker
ThreadFromGlobalPool::init()::{lambda()#1}::operator()
std::__1::__thread_proxy
start_thread
clickhouse-serv
root

典型问题排查场景提效

秒级聚合分析

持续Profiling天然支持按照每秒进行分析，能够看出很多直接用单张聚合后的火焰图无法看出的细微抖动问题.

线上偶发抖动

现有手段一般为根据特征去写脚本,出现后挂perf采集火焰图.
持续Profiling只要开启即一直采集，随时可以看任意时间的抖动问题，无需费力抓现场

进程启动调试

进程启动过程中用perf记录依赖pid，而pid只有启动后才会分配.
一般需要手动加sleep去进行采集.
持续Profiling一直都在采集，无需关心启动pid.

实现方案详解

核心问题：解析库的内存占用

我们主要做了两个工作来优化全局profiling的内存占用：

elf_ctx/pid_ctx分层

- elf_ctx存储debuginfo/symbol. pid_ctx存储映射信息和涉及elf_ctx的列表
- elf_ctx用引用计数管理，多个进程引用同一个elf，内存中仅有一份实体. 类似共享库的思想.

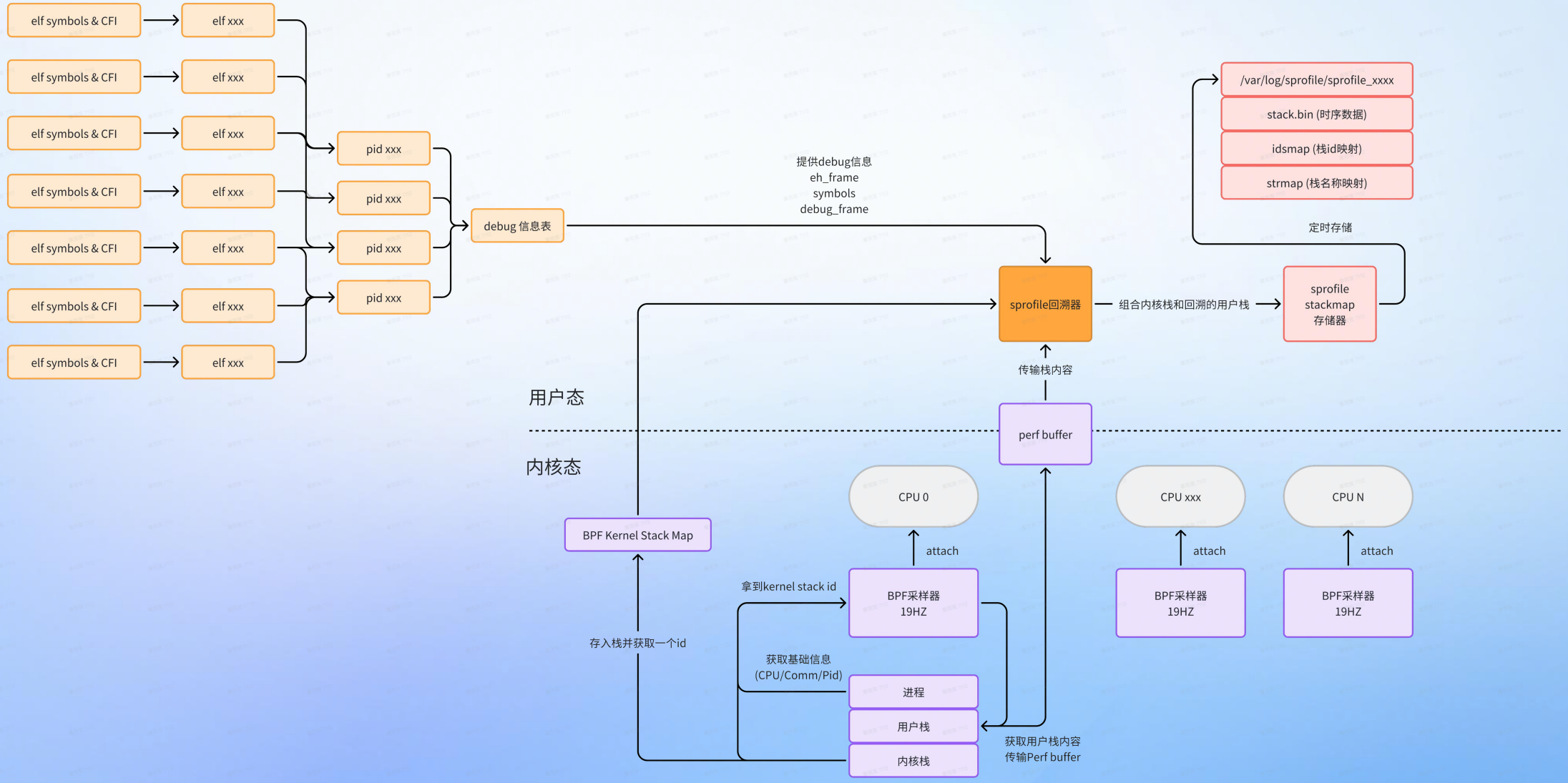
debuginfo精简

- 仅保留symbol和CFI信息，去除所有例如debug_lines等section
- CFI原始信息和symbols均在初始化中全加载到内存. CFI信息在解析过程中展开为表达式，不做缓存.

基于左侧所述工作：

- 极大缩减debuginfo整体的内存占用.
- CPU解析速率稳定高效. debuginfo全部在内存，无磁盘缓存层.

实现架构


elf symbols & CFI
elf xxx
pid xxx
debug 信息表
提供debug信息
eh_frame
symbols
debug_frame
/var/log/sprofile/sprofile_xxxx
stack.bin (时序数据)
idsmap (栈id映射)
strmap (栈名称映射)
定时存储
sprofile回溯器
组合内核栈和回溯的用户栈
sprofile
stackmap
存储器
传输栈内容
用户态
perf buffer
内核态
CPU 0
CPU xxx
CPU N
BPF Kernel Stack Map
attach
拿到kernel stack id
BPF采样器
19HZ
存入栈并获取一个id
获取基础信息
(CPU/Comm/Pid)
进程
用户栈
内核栈
获取用户栈内容
传输Perf buffer

现状和开源计划

字节内部使用现状

已大批量铺开，在多个业务全量使用．广泛应用到各类问题排查场景．并借助字节的业务规模，经过半年以上的解析案例迭代，将栈解析准确率提升到和Perf相当甚至某些场景更优的水平．

开源计划

我们预计明年年初会将回溯库，持续profiling程序 和 解析SDK 开源．
并尝试是否可以为bcc / bpftrace等常见bpf工具提供解析ustack能力．

Q&A

THANKS