第二届 eBPF开发者大会

www.ebpftravel.com

eBPF揭开可观测性盲区实现根因推导

杭州云观秋毫 —— 苌程

Kindling-Originx全球首创的故障根因推理引擎

中国·西安

1、自我介绍

苌程

2010年浙江大学SEL实验室助理研究员工作10年，专注在云原生和分布式计算领域

2016年作为技术联合创始人创立杭州谐云科技，专注在容器云

2022年基于eBPF的Kindling开源项目创始人
https://github.com/KindlingProject/kindling

2023年作为CEO创立杭州云观秋毫科技有限公司，专注在故障根因定位

2024 年1月推出全球首创的故障根因推理引擎

专家的故障排查之道

现象一：XX延时升高

现象二：XX报错

现象三：网络在XX时刻出现抖动

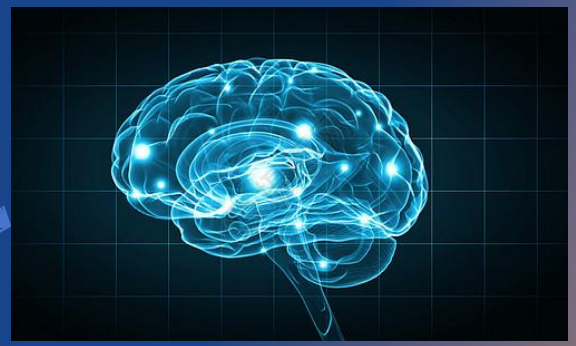
现象四：日志报XXX错

可能性一XXX

可能性二XXX

可能性三XXX

……



经验丰富的专家

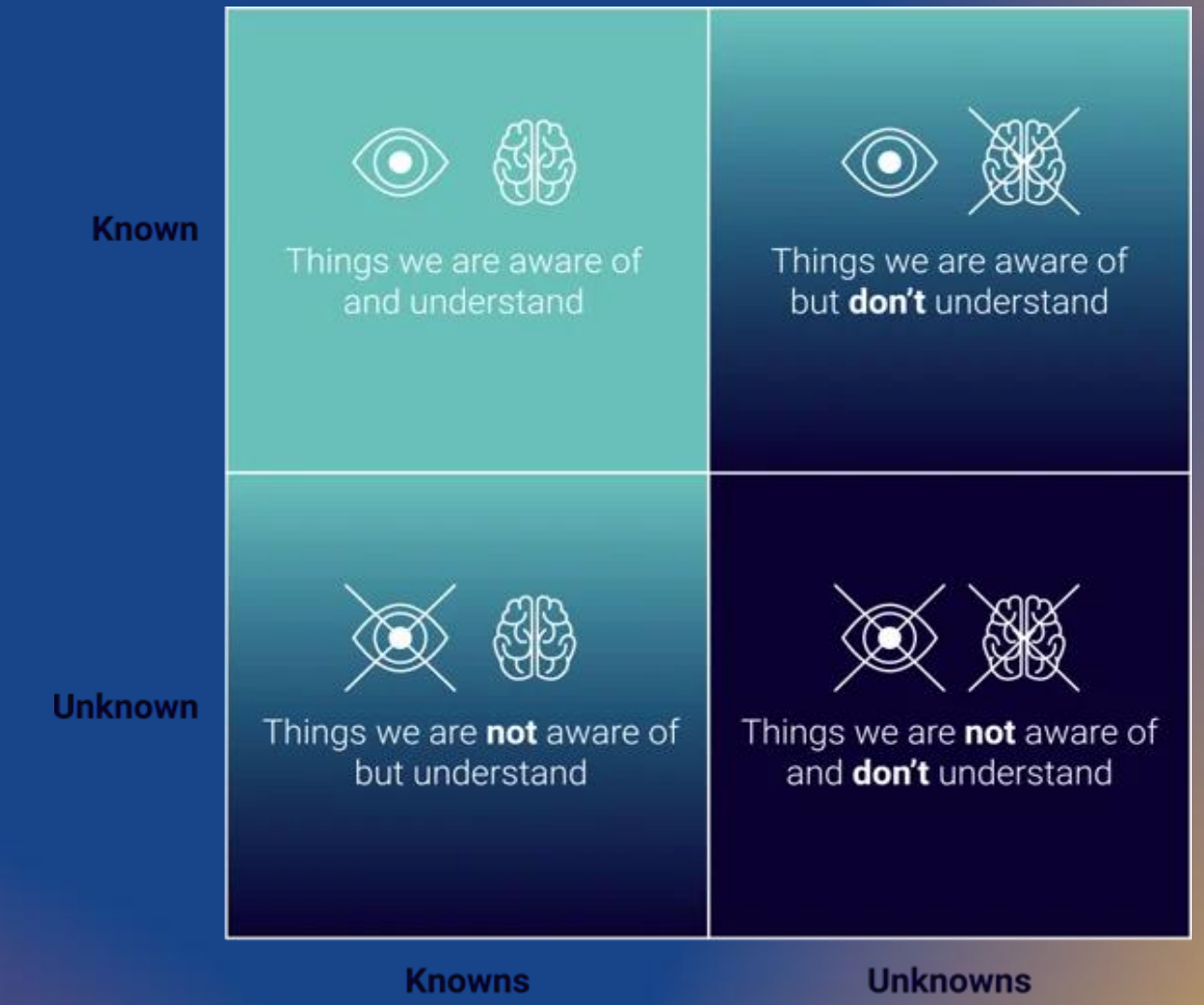
问题的源点——专家为什么要使用启发式的排障方式

如果我们知道发生了什么
找到根因是容易的

可观测性工具的出现
帮助我们理解程序执行过程中发生了什么

在可观测性性工具的帮助之下
仍然有很多盲区

盲区的存在，
导致人对故障根因的认识都是有限的



Gartner对于盲区问题的观点
启发式排障解决盲区Unknown-Unknown问题

The ability to perform interactive exploration and analysis of multiple telemetry types (such as traces, metrics, logs) to detect “unknown unknowns”

专家利用现有工具都需要灵感启发才能定位故障根因
对于AIOPS，难以提供准确故障根因标记数据

过度依赖于专家带来的问题

技术能力的不均衡，专家过于忙碌：团队内技能水平的差异导致高级工具和数据的利用率低下。

知识传递的困难： 缺乏有效的机制将高级用户的经验和知识快速传递给新手或非专家用户。

故障响应的时效性： 在发生故障时，需要快速有效的响应，但技术水平不一致可能导致延迟。

培训和提升的成本： 提升团队整体技术水平需要大量的时间和资源投入。

2、基于eBPF实践TSA方法论完整还原程序执行过程——开源Kindling Trace-Profiling打开盲区

The TSA Method：
https://www.brendangregg.com/tsamethod.html#:~:text=The%20TSA%20Method%20is%20a,an%20initial%20handle%20on%20performance.

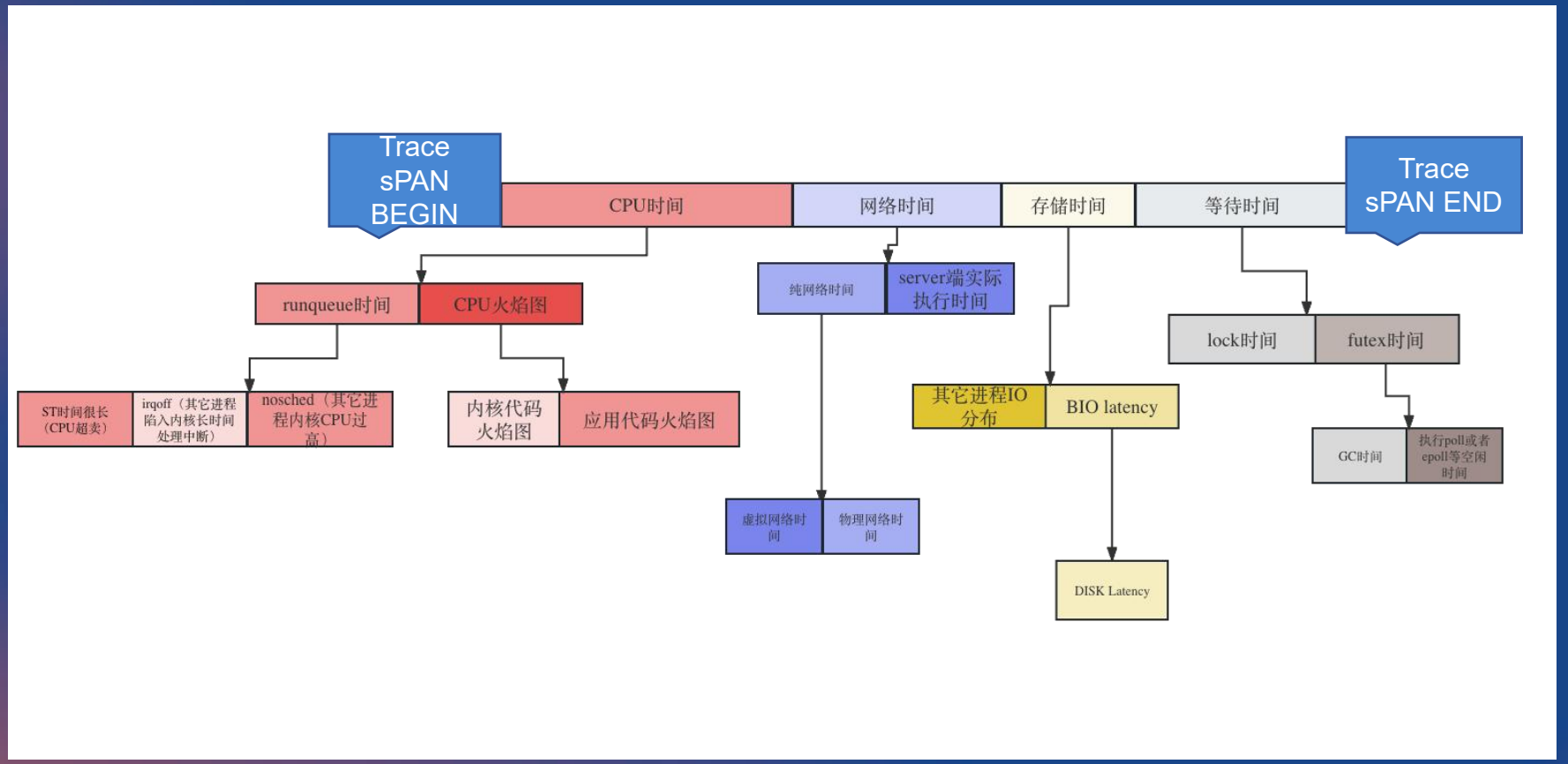
仍然面临的问题：

内核事件与系统调用接受度差：业务开发人员看不懂内核事件与系统调用

变相依赖专家的另一种能力：从依赖排障专家变成依赖于内核事件专家

技术落地难：完整的程序执行过程数据量庞大，难以在生产环境落地

基于排障北极星指标来完成盲区的覆盖



完备

全覆盖

构建请求的北极星指标基线——很容易识别一次请求的局部异常

接口执行耗时分析

历史基线

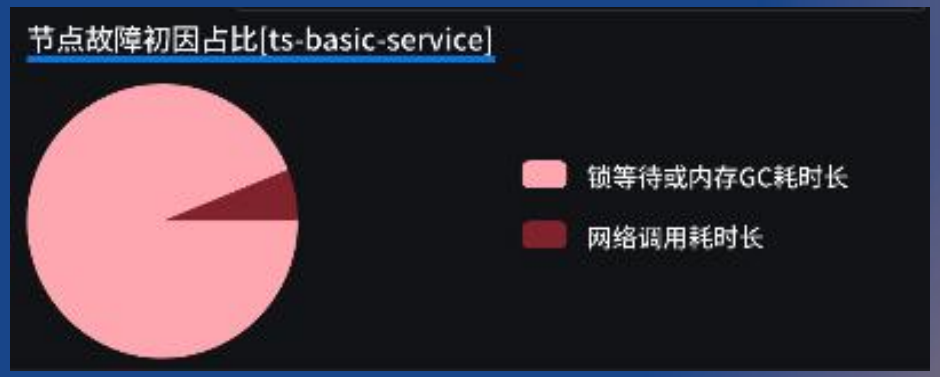
接口执行耗时分析

	历史基线	本次耗时
cpu	6.61ms	7.81ms
file	0.77ms	1.63s
net	4.05ms	1.43ms
futex	0.63ms	0.17ms
other	0.04ms	9.63ms
epoll	3.07ms	0.49ms
runq	3.51ms	1.33ms

局部异常的统计分析作为故障的初因判定

单次请求的局部异常可能是偶发因素

对局部异常进行统计分析可以进一步判定故障节点的局部异常是否为故障根因，准确度大幅提高



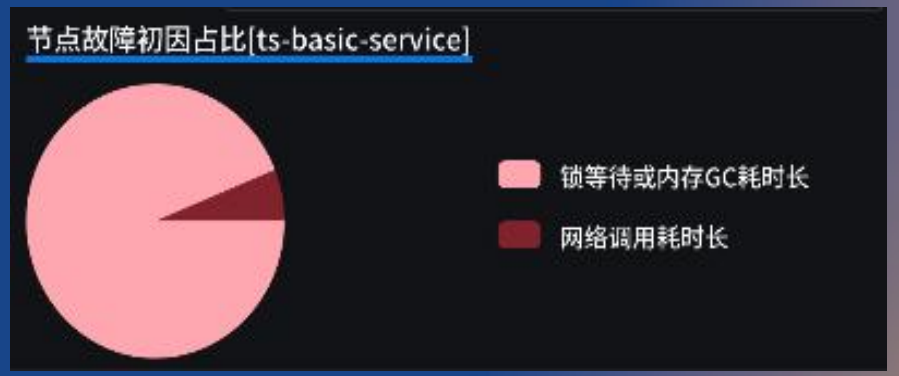
出现这个特征的疑似节点
大概率不是根因节点

利用算法对全局故障节点做疑似节点的筛查——完成根因的定位

根据各种论文模型对疑似节点进行筛查

- MicroRank
- TraceRCA
- GTrace
- TraceVAE
- CRISP
- TraceAnomaly
- Trace Analysis for Microservice Architecture

疑似节点准确度不是最重要的，允许误报情况



局部根因统计能够对算法结论进行进一步根因的确定，所以能允许误报

3、故障推理引擎Originx对eBPF能力的理解

——以内核可观测性数据（程序行为建模）关联其它可观测性数据

通过eBPF内核可观测性数据对程序行为建模：穿针引线联动各种可观测性数据

- 应用可观测性数据
- 网络可观测性数据
- 日志可观测性数据
- 各种事件

规避了构建运维数据大集中平台的大投入

“穿针引线”可观测性数据

CPU异常——代码执行效率	RUNQ异常——CPU资源紧张	NET异常——疑似网络问题	File异常——疑似存储问题	Futex异常——业务锁
代码执行堆栈	CPU利用率	网络耗时发生在Client、Server、传输三者一侧	IOPS	程序GC判断
CPU Throttle	网络重传和网络延时	节点读磁盘指标	锁的代码堆栈	
CPU Steal	网络带宽和节点的CPU利用率	节点写磁盘指标		

接口执行耗时分析

	历史基线	本次耗时
cpu	4.73ms	3.63ms
net	2.83ms	74.42ms
futex	0.56ms	114.40ms
other	0.03ms	0.01ms
runq	0.95ms	12.53ms

4、北极星指标异常的告警 vs 传统基于指标告警

基于北极星指标告警

- 对程序执行过程异常告警
  - 阀值设定简单易懂
- 异常与根因关系非常清晰
  - 直接引导排障方向
- 告警风暴可以规避

VS

传统基于指标告警

- 对程序执行结果告警
  - 阀值设定困难
- 告警与根因关系不明显
  - 信息不足
- 告警风暴

案例说明一

Node的CPU利用率60%该告警吗?

- 具体阀值很难合理设置（业务IO密集型和CPU密集型有关）

VS

北极星指标CPU相关异常

- CPU异常——代码循环嵌套比以往多
- RUNQ异常——节点的CPU资源不足（不管其CPU使用率）或者容器的Limit设置不合理

案例说明二

网络监控发现重传指标波动告警

- 业务现在抖动大概率是由于重传指标波动告警导致的

VS

北极星指标网络也许没有告警

- 网络重传并不一定会导致程序执行出现异常

案例说明三

应用监控反应出某节点执行时间偏差最大

- 排查日志
- 排查代码执行情况
- 结论这个节点为什么异常？不一定能给出结论

VS

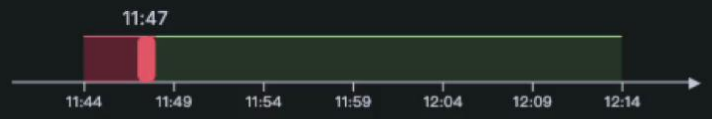
北极星指标告知网络调用偏差很大

- 网络异常
- Futex等待异常
- 结论：大概率该节点并不是根因

服务健康AI检测诊断

SLO实时异常检测　SLO近48小时异常检测

服务入口 Enter value　排序 Choose　异常聚焦　5m

服务入口	近30分钟	异常诊断	异常跟节点占比	操作
查询余票		2024年4月8日 11:47 - 11:48 请求成功率 = 100% SLO可用性目标是95 % P90 = 12001.33ms SLO延迟性目标是645.23 ms	ts-travel-service 81.82% 	详情 配置 诊断
查询最便宜的票		2024年4月8日 12:13 - 12:14 请求成功率 = 100% SLO可用性目标是95 % P90 = 1123.64ms SLO延迟性目标是1000.01 ms	ts-order-service 33.33% 	详情 配置