AI在内核故障定位的应用实践

Author: 陈增溪 chenzengxi@huawei.com

CONTENTS

HUAWEI

背景介绍 – 数量多，种类杂

- 虚拟化运维面临的问题
  - 线上/线下百万量级的服务器
  - 内核可用性<99.99%
  - 日均问题数量 1,000,000 × 0.01% = **100**

- 常见的问题种类
  - 宕机类 – 硬件失效/MCE/内核/驱动bug/死锁/OOM，etc.
  - 性能类 – sys或软中断冲高/IO时延增大/网络抖动
  - 存储类 – 文件系统无法挂载/只读/设备发现失败
  - 网络类 – 网络不可达/丢包/时延
  - …

日志分析 -> 根因定位 -> 规避方案 -> 解决方案

背景介绍 – 定位难，耗时长

- **Panic内核故障定位过程**
  - 异常堆栈分析
  - 反汇编，参数推导
  - 源码分析
  - 故障原因推测，故障注入/复现
  - 方案测试验证

```
 #7 [ffff9a62a0883d08] __do_page_fault at ffffffffab52d6a0
 #8 [ffff9a62a0883d70] do_page_fault at ffffffffab52d885
 #9 [ffff9a62a0883da0] page_fault at ffffffffab529768
    [exception RIP: rcu_check_callbacks+467]
    RIP: ffffffffaaf530d3  RSP: ffff9a62a0883e50  RFLAGS: 00010046
    RAX: 0000000000000000  RBX: ffff9a624c316220  RCX: ffffffffaba8aba0
    RDX: 0000000000000164  RSI: ffff9a624c316220  RDI: ffffffffaba4a678
    RBP: ffff9a62a0883ea8   R8: 0000000000000000   R9: 0000000000000001
    R10: 0000000000000164  R11: ffff9a1cc602af70  R12: ffffffffaba4a678
    R13: 0000000000000012  R14: 0000000000000000  R15: ffff9a1cc602af70
    ORIG_RAX: ffffffffffffffff  CS: 0010  SS: 0018
#10 [ffff9a62a0883e60] __wake_up_common at ffffffffaaec967b
#11 [ffff9a62a0883eb0] update_process_times at ffffffffaaea8ff6
#12 [ffff9a62a0883ed8] tick_sched_handle at ffffffffaaf07ee0
#13 [ffff9a62a0883ef8] tick_sched_timer at ffffffffaaf08119
#14 [ffff9a62a0883f20] __hrtimer_run_queues at ffffffffaaec3bd3
#15 [ffff9a62a0883f78] hrtimer_interrupt at ffffffffaaec415f
```

```
crash> dis rcu_check_callbacks
0xffffffffaaf52f00 <rcu_check_callbacks>:       nopl   0x0(%rax,%rax,1)
0xffffffffaaf52f05 <rcu_check_callbacks+5>:     push   %rbp
0xffffffffaaf52f06 <rcu_check_callbacks+6>:     mov    %rsp,%rbp
0xffffffffaaf52f09 <rcu_check_callbacks+9>:     push   %r15
0xffffffffaaf52f0b <rcu_check_callbacks+11>:    push   %r14
0xffffffffaaf52f0d <rcu_check_callbacks+13>:    mov    %edi,%r14d
0xffffffffaaf52f10 <rcu_check_callbacks+16>:    push   %r13
0xffffffffaaf52f12 <rcu_check_callbacks+18>:    push   %r12
0xffffffffaaf52f14 <rcu_check_callbacks+20>:    mov    %esi,%r12d
0xffffffffaaf52f17 <rcu_check_callbacks+23>:    push   %rbx
```

专业性强，难度大，耗时长

每次宝贵的定位经验不该被浪费

CONTENTS

HUAWEI

青鸟 – 故障自动诊断系统

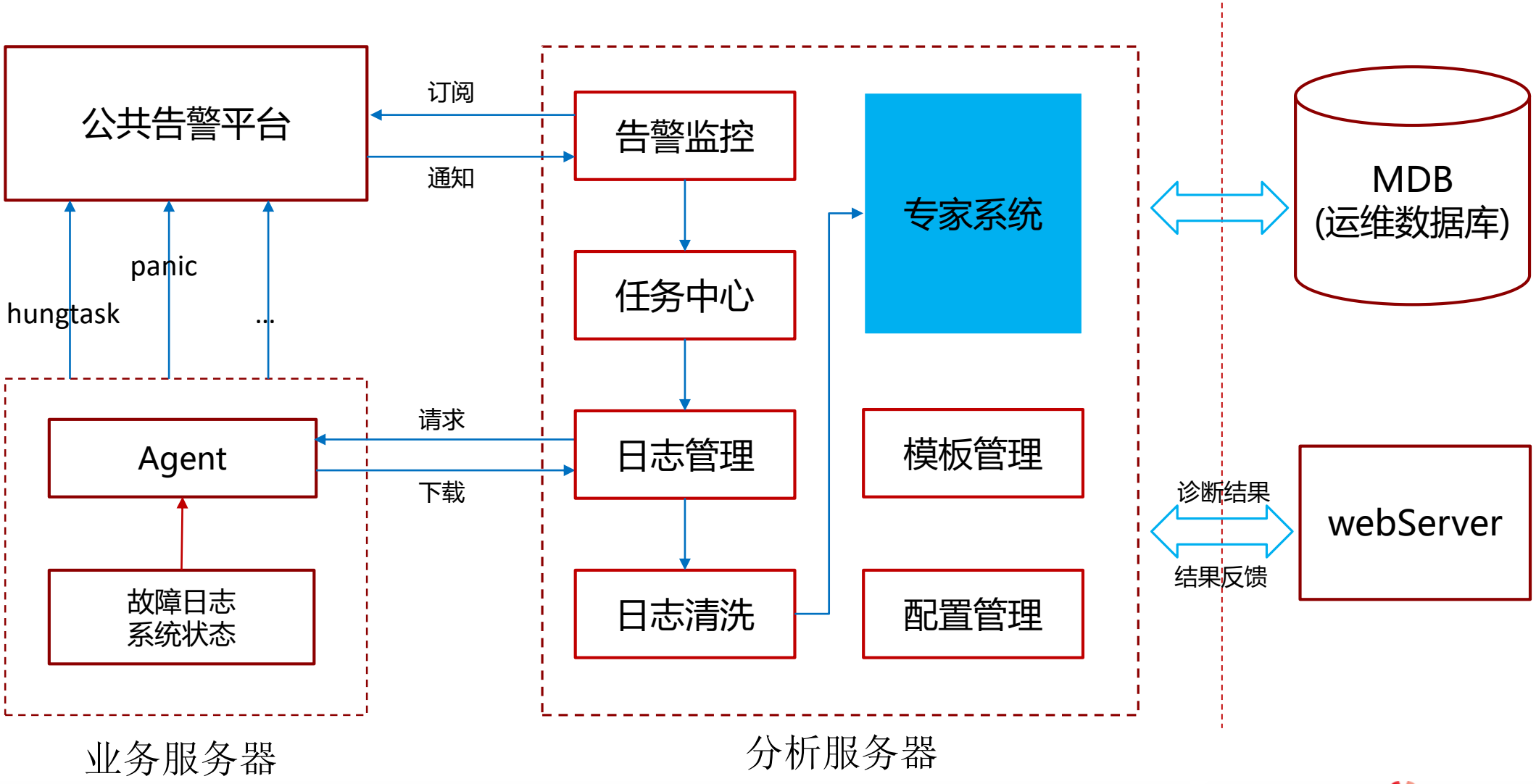
AS-IS

1. 人工分析
2. 专业性强，耗时长
3. 重复分析
4. 被动跟进

TO-BE

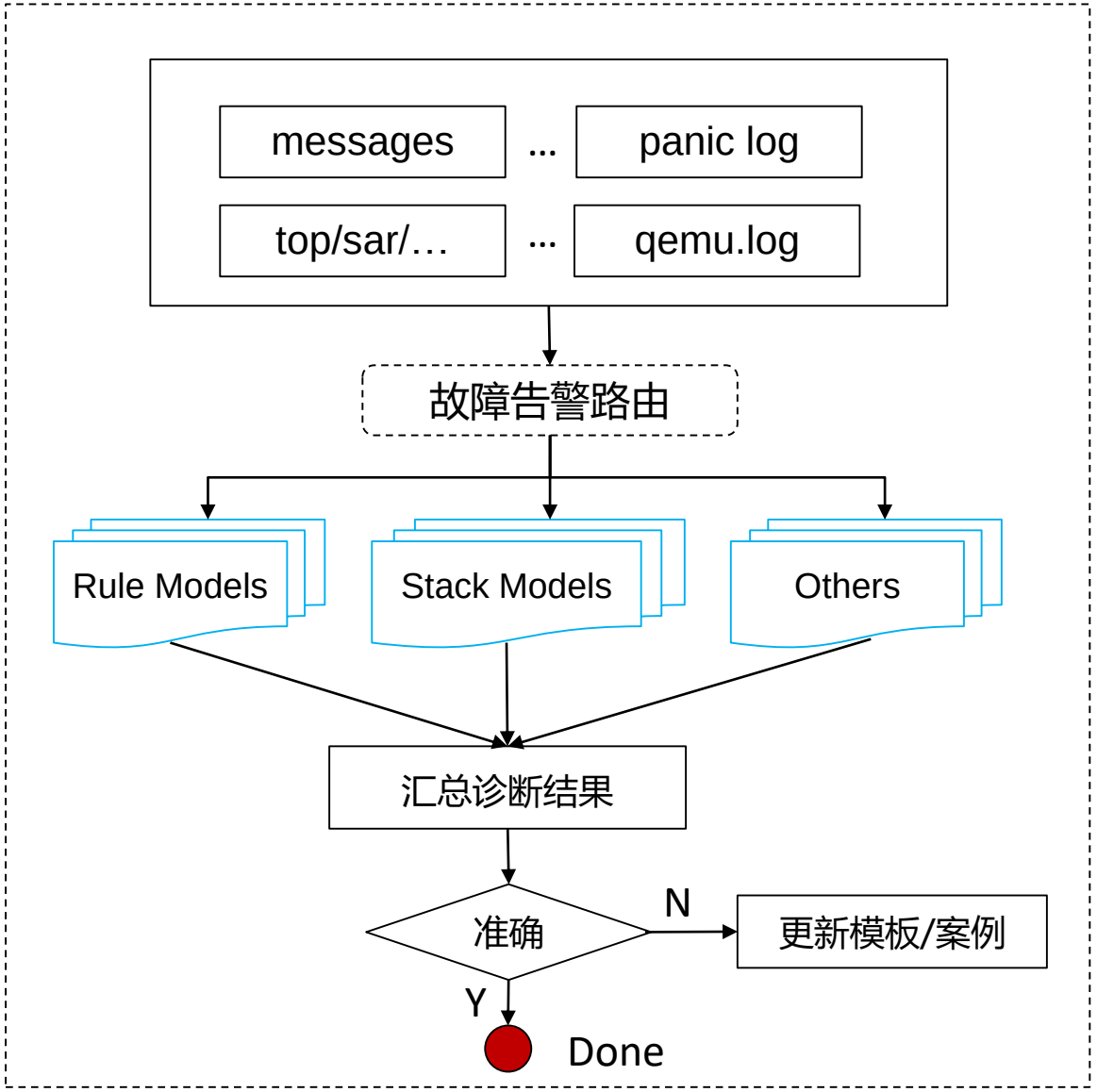
1. 自动分析
2. 简单自助，分钟量级
3. 一次就好
4. 主动发现

青鸟 – 故障自动诊断系统


公共告警平台
订阅
通知
告警监控
专家系统
MDB
(运维数据库)
任务中心
panic
hungtask
...
请求
Agent
日志管理
模板管理
下载
诊断结果
webServer
结果反馈
故障日志
系统状态
日志清洗
配置管理
业务服务器
分析服务器

青鸟 – 故障自动诊断系统

- **异常检测 – 识别已知的故障模式**
  - 支持按日志模板匹配进行异常识别
  - 支持自定义脚本根据日志上下文进行异常诊断
- **堆栈分析 – 识别已知的宕机类问题**
  - 日志堆栈信息识别、提取
  - 已知相似案例搜索 – **关键：衡量堆栈的相似度**
- **状态分析 – 辅助故障定位**
  - 性能日志分析
  - 系统状态分析
  - 虚机生命周期分析
  - …
- **模板管理 – 学习反馈路径**
  - 日志模板的更新
  - 案例模板的更新



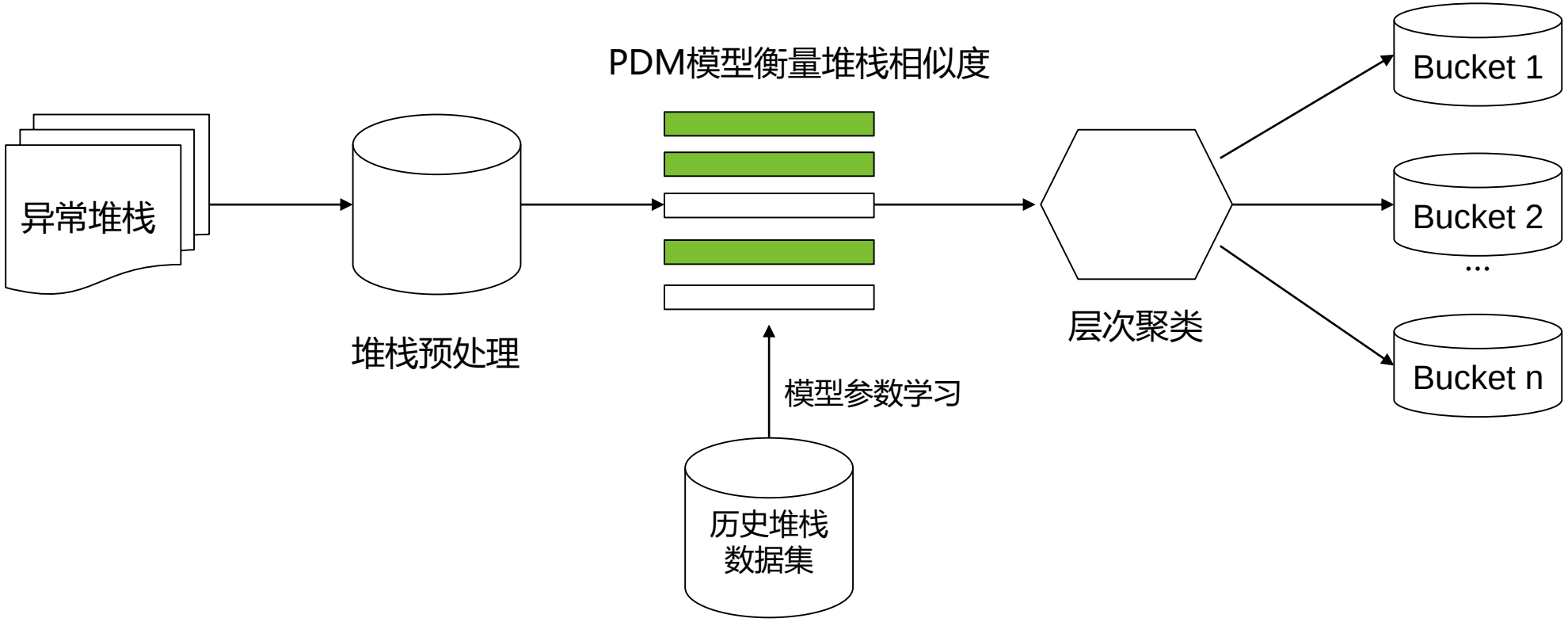
CONTENTS

HUAWEI

关键算法介绍

总体流程

- 堆栈预处理 – 函数白名单
- 使用**PDM模型**衡量堆栈的相似度，其中模型参数通过**历史堆栈数据集**机器学习获取
- 通过层次聚类方法将相似的堆栈聚类到相应的Bucket内



堆栈数据集

数据源

- 总结华为云历史宕机类/软硬死锁/D状态类问题，提取合法堆栈
- 爬取kernel社区中log msg中包含Call Trace的问题，进一步人工筛选

xfs: log head and tail aren't reliable during shutdown

```
I'm seeing assert failures from xlog_space_left() after a shutdown
has begun that look like:

XFS (dm-0): log I/O error -5
XFS (dm-0): xfs_do_force_shutdown(0x2) called from line 1338 of file
XFS (dm-0): Log I/O Error Detected.
XFS (dm-0): Shutting down filesystem. Please unmount the filesystem
XFS (dm-0): xlog_space_left: head behind tail
XFS (dm-0):   tail_cycle = 6, tail_bytes = 2706944
XFS (dm-0):   GH   cycle = 6, GH   bytes = 1633867
XFS: Assertion failed: 0, file: fs/xfs/xfs_log.c, line: 1310
------------[ cut here ]------------
Call Trace:
 xlog_space_left+0xc3/0x110
 xlog_grant_push_threshold+0x3f/0xf0
 xlog_grant_push_ail+0x12/0x40
 xfs_log_reserve+0xd2/0x270
 ? __might_sleep+0x4b/0x80
 xfs_trans_reserve+0x18b/0x260
```

数据集格式

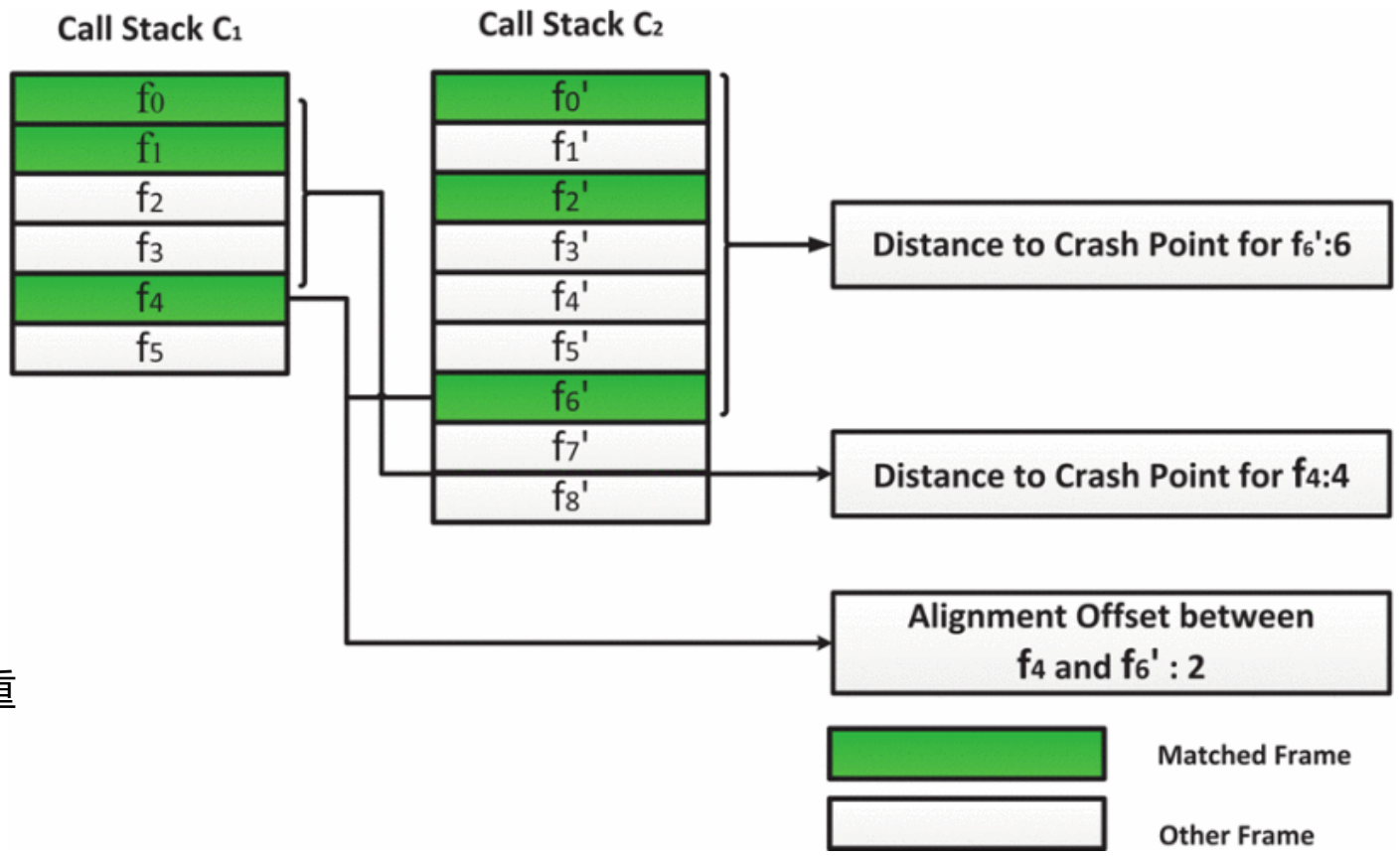

```
⊟[
  ⊟{
      "stack_id":1,
      "duplicate_stack":"",
      "symbols":⊟[
          "__mutex_lock_slowpath",
          "mutex_lock",
          "do_lookup",
          "do_last",
          "path_openat",
          "do_filp_open",
          "do_sys_open",
          "system_call_fastpath"
      ]
  },
  ⊟{
      "stack_id":2,
      "duplicate_stack":"",
      "symbols":⊟[
          "dump_trace",
          "dump_stack",
          "warn_slowpath_common",
          "warn_slowpath_fmt",
          "tcp_recvmsg",
```

PDM模型

Position Dependent Model

位置相关模型

- **两个关键度量**
  - Distance to the Top Frame
  - Alignment Offset
- **基于两个假设**
  - 函数离栈顶越近，应分配更多权重
  - 相似堆栈的对齐偏移很小



PDM模型

$$L = \{L_1, L_s, L_3 \ldots\} \; L_i = \{S_{i,1}, S_{i,2}, S_{i,3}, \ldots S_{i,k} \ldots\}$$

$$\begin{cases} sim(C_1, C_2) = \dfrac{\max\limits_{L_i \in L}[Q(L_i)]}{\sum\limits_{j=0}^{l} e^{cj}} \\ Q(L_i) = \sum\limits_{s_{i,k} \in L_i} e^{-c\min(Pos(C_1 s_{i,k}), Pos(C_2 s_{i,k}))} e^{-o|Pos(C_1 s_{i,k}) - Pos(C_2 s_{i,k})|} \end{cases} \quad (1)$$

$$M_{i,j} = \max \begin{cases} M_{i-1,j-1} + \cos t(i,j) \\ M_{i-1,j} \\ M_{i,j-1} \end{cases} \quad (2)$$

$$\cos t(i,j) = \begin{cases} e^{-c^* \min(i,j)} e^{-o^* obs(i-j)} & if\ ith\ frame of\ C_1 = jth\ frame\ of\ C_2 \\ 0 & otherwise \end{cases} \quad (3)$$

$$sim(C_1, C_2) = \frac{M_{m,n}}{\sum_{j=0}^{l} e^{-cj}} \quad (4)$$

- L定义为C_1, C_2的公共子序列集合，S定义为L_i内相匹配的函数
- $Pos(C_i, S_{i,k})$定义为函数在堆栈内的位置
- l为堆栈C_1, C_2中函数数量的最小值
- 指数函数参数c, o分别是到栈顶的距离系数和对齐偏移系数
- $Q(L_i)$用来衡量的函数到栈公共帧序列中匹配的函数的相似度值，其中第一个指数函数考虑了一对匹配顶的最小距离，第二个指数函数考虑最小对齐偏移

- 由公式可知，堆栈相似度由最大公共帧序列决定可用二维动态规划方法优化求解降低计算复杂度参考公式(2,3,4)

项目中的算法实现

- 计算两个堆栈距离
  - 设定c，o参数
  - 分别获取两个堆栈长度
  - M二维矩阵保存动态规划过程中各层级的相似度计算结果
  - sum存储较短堆栈的计算结果
  - 按公式计算相似度
  - 返回距离

- 训练态时定义堆栈距离大于0.08的为不同的聚类
- 实际应用时可放宽到0.2，小于0.2时两个堆栈已非常相似

```python
def get_distance(stack1, stack2):
    """
    计算两个堆栈的距离，结果范围(0.0 ~ 1.0). 1.0：完全不同，0.0：完全相同
    :param stack1: 堆栈1列表
    :param stack2: 堆栈2列表
    :return: 堆栈距离
    """
    # c和o的系数取值由机器学习建模训练获得
    c = [redacted]  # 当前函数到栈顶距离系数(Distance to the Top Frame)
    o = [redacted]  # stack1, stack2中相同函数的对齐偏移系数(Alignment Offset)
    len1 = len(stack1)
    len2 = len(stack2)

    if len1 == 1 or len2 == 1:
        return 1.0

    m = [[0 for col in range(len2 + 2)] for row in range(len1 + 2)]
    for i in range(1, len1 + 1):
        for j in range(1, len2 + 1):
            x = 0
            if stack1[i - 1] == stack2[j - 1]:
                x = math.exp(-c * min(i - 1, j - 1)) * math.exp(-o * math.fabs(i - j))
            m[i][j] = max(max(m[i - 1][j - 1] + x, m[i - 1][j]), m[i][j - 1])

    _sum = 0
    for i in range(min(len1, len2)):
        _sum += math.exp(-c * i)

    similarity = m[len1][len2] / _sum if _sum != 0 else 0

    return 1.0 - similarity
```

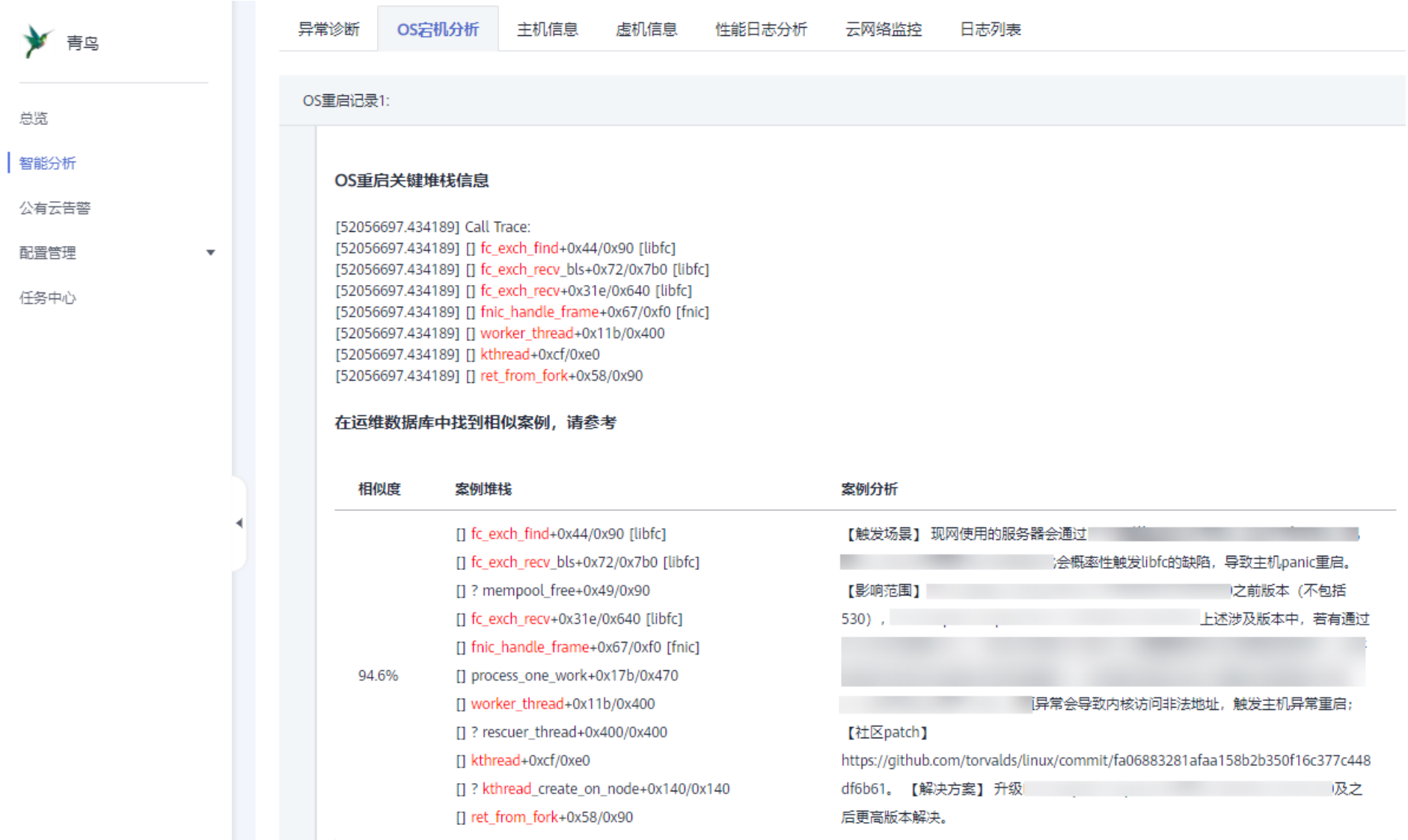
CONTENTS

HUAWEI

举例演示

案例1

根据重启的异常日志堆栈分析后，找到相似度为94.6%的已知案例。运维人员结合故障场景判定为同一问题。快速给出问题根因及解决方案。

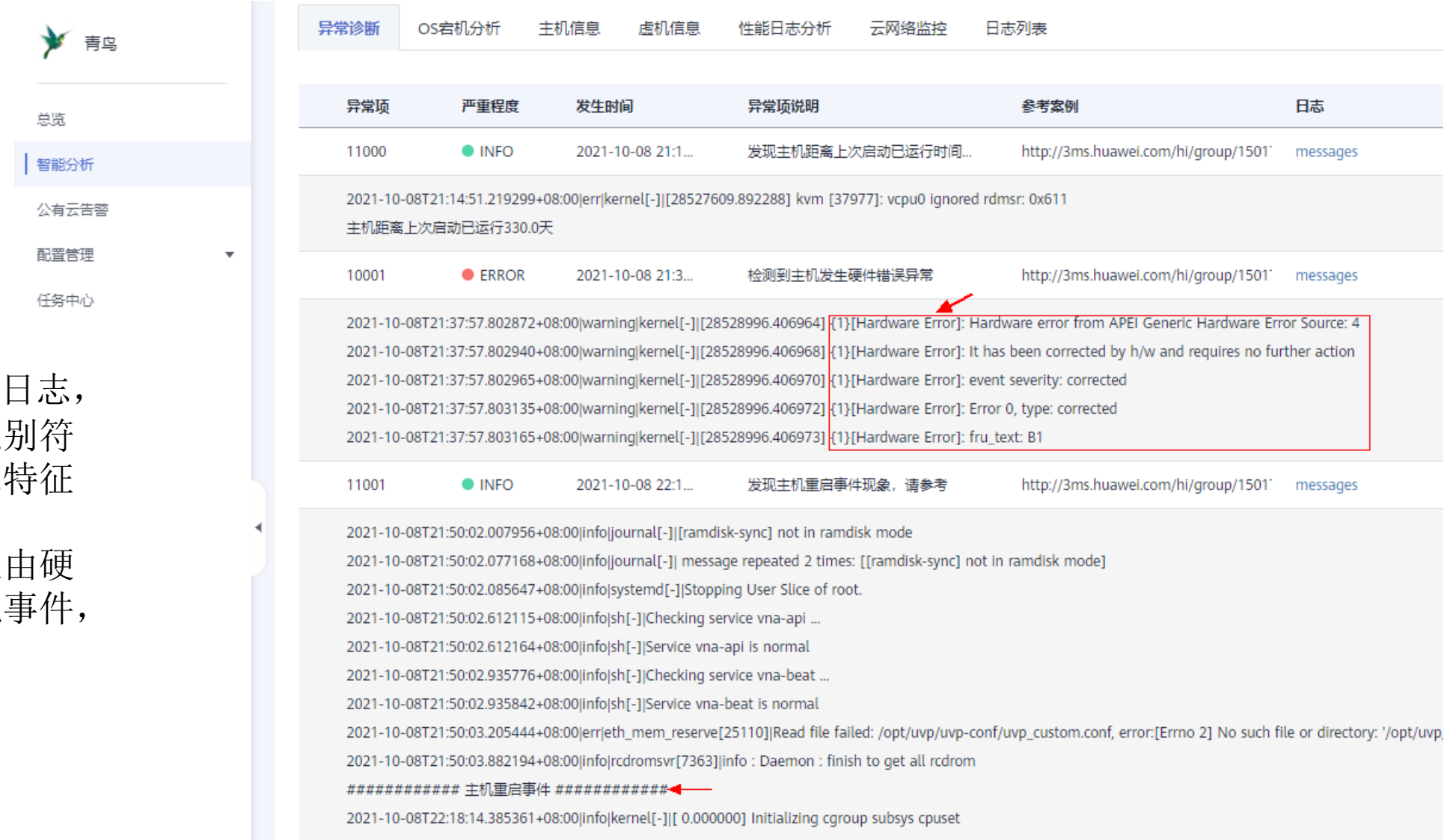


举例演示

案例2

根据自动分析messages中的异常D状态堆栈，可以从运维知识库获取对本次D状态发生的场景、内核流程、可能性原因等。辅助运维人员对问题做进一步分析。

举例演示



案例3

自动分析messages日志，根据日志模板，识别符合硬件故障的日志特征(Hardware Error)。
根据时序判断这是由硬件故障导致的宕机事件，提单硬件解决。

小结

- 已知问题基于日志自动化分析
- 未知问题根据日志做初步辅助分析
- 未知问题完成分析后反馈系统，未知问题 -> 已知问题

下一步工作

- 持续扩充运维知识库，全面覆盖现有已知问题 → 已知问题全自动化
- 未知问题，提供更全面的辅助分析信息 → 未知问题半自动化
- 更高的异常检测效率
- 基于可靠的根因诊断尝试故障自动修复

Thank you

www.huawei.com