

赛道一：基于大模型智能体的微服务根因定位

# IMARS: Intelligent Multi-modal Anomaly & Root- cause System 智能多模态异常与根因分析系统

北京邮电大学

网络与交换技术全国重点实验室 网络管理研究中心

郭鑫、张宇轩、鲁奕文、于家傲 指导教师：芮兰兰

小鸟吃大蒜(蒜鸟)

主办单位：中国计算机学会（CCF）

承办单位：中国计算机学会互联网专委会、中国科学院计算机网络信息中心、中国移动研究院、清华大学

协办单位：华为2012实验室、阿里云、中兴通讯、中国移动九天团队、南开大学、西安电子科技大学、清华大学计算机科学与技术系、神州灵云



北京邮电大学



网络与交换技术全国重点实验室  
State Key Laboratory of Networking and Switching Technology

# 目录 CONTENTS

第一章节 赛题分析

第二章节 方案介绍

第三章节 总结

# 第一章节

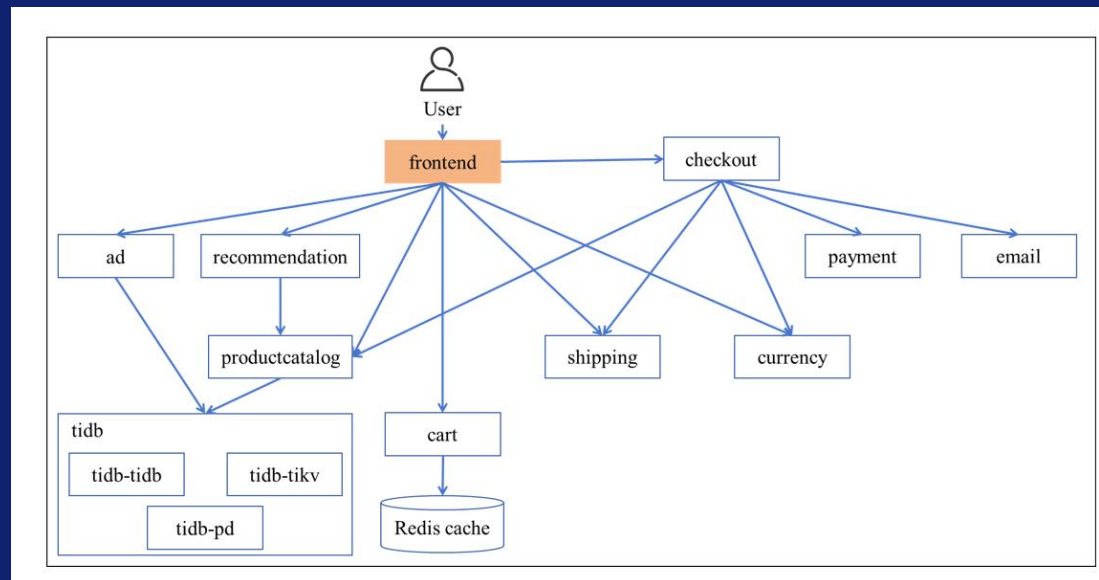
## 赛题分析

本赛道聚焦于微服务系统中的根因定位问题。面对服务依赖复杂、监控数据多样的挑战，传统方法难以高效准确定位故障根因。基于比赛提供的微服务系统数据，构建一个融合监控指标、日志和调用链数据的大模型根因定位系统，实现自然语言理解、智能分析，并自动识别异常的根因组件与故障类型，推动智能体技术在实际运维场景中的应用。

挑战一：如何高效利用海量数据？

挑战二：如何有效准确的进行根因定位？

挑战三：如何保证实际应用可行性？



## 第二章节

# 方案介绍

# 方案介绍

## 结构化输出

定位异常组件  
精准定位故障点

根因解释  
详细原因分析

推理过程  
详细推理步骤

自动化流程  
端到端无人干预

## 高并发处理

多进程并发池  
Process Pool Executor

异步队列机制  
Manager().Queue.()

锁机制  
防止重复处理

实时日志追踪  
全程监控与调试

## AI驱动的根本分析

LLM推理  
基座模型更换  
多维度数据融合

智能推理链  
因果关系分析  
依赖链追踪

专家规则  
阈值判断规则  
故障分类逻辑

结果生成  
组件定位  
故障影响分析  
原因解释

## 多维度异常检测

Service级异常  
LSTM+3 $\sigma$ 检测  
rrt/rrt\_max

Pod级异常  
Pod实例监控  
细粒度检测

Pod资源异常  
CPU/内存/进程  
阈值规则

Node资源异常  
VM资源监控  
时序差值分析

异常log  
87个关键词  
时间关联

Trace分析  
状态码  
耗时

## 数据输入

数据预处理

时间区间解析

多维度融合

log

metric  
-apm-  
service

metric  
-apm-  
pod

metric  
-infra-  
node

metric  
-  
infra-  
pod

metric  
-  
infra-  
tidb

metric  
-  
infra-  
pd

metric  
-  
infra-  
tikv

trace



# 方案介绍

处理链路：数据预处理 -> 读取数据 -> 异常检测 -> AI驱动的根本因分析 -> 结构化输出

```
{  
  "Anomaly Description": "The system experienced an anomaly from 2025-06-05T16:10:02Z to 2025-06-05T16:31:02Z. Please infer the possible cause.",  
  "uuid": "345fbe93-80 "  
}
```

① 提取时间：正则匹配 或者 小模型

Start time: 2025-06-05T16:10:02Z  
End time: 2025-06-05T16:31:02Z

② 读取数据&异常检测

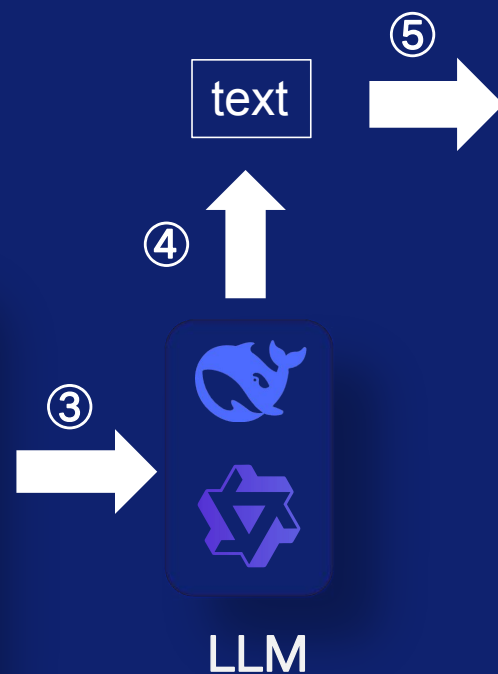


Data

+



Prompt

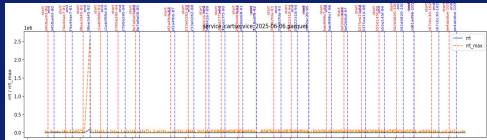


```
{  
  "component": "emailservice",  
  "uuid": "345fbe93-80",  
  "reason": "CPU stress; pod_cpu_usage=0.4, pod_processes=10",  
  "reasoning_trace":  
    [{  
      "step": 1, "action": "Analyze service metrics",  
      "observation": "emailservice shows abnormal_rrt (max_error=27869.24) and abnormal_rrt_max (max_error=191644.45)"  
    },  
    {  
      "step": 2, "action": "Inspect pod metrics",  
      "observation": "All emailservice pods have persistent pod_cpu_usage=0.4 (>0.2) and pod_processes=10 (>1 threshold)"  
    },  
    {  
      "step": 3, "action": "Verify trace data",  
      "observation": "emailservice self-calls (e.g., SendOrderConfirmation) show high duration (98ms) but no external dependencies"  
    }  
  ]  
}
```

# 挑战一：如何高效利用海量数据？

针对 5 类数据，分别设计并采取了不同的异常检测方法，显著提高了异常数据利用率

① RRT



metric-apm-pod

metric-apm-service

rrt rrt\_max

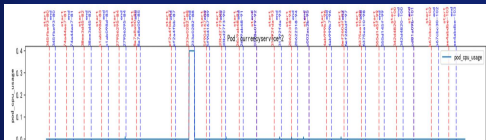
LSTM预测误差 + 3std

service/pod max\_error  
top3\_mean threshold  
max\_error\_detail  
anomaly\_count

error time out

>0 即可能为异常

② Infra



metric-infra-pod

metric-infra-node

cpu\_usage\_rate

memory\_usage\_rate

filesystem\_usage\_rate

pod\_processes

...

规则手动配置，易于修改和扩展

③ log

命中关键词的日志记录可能为异常

"error","exception","failed","fail","failure","abort","panic","critical",  
"overflow","underflow",  
"hang","stall","block","pause","deadlock","jam","stop","GC",  
"retry","reset","revoked","suppressed","ignored","unresponsive"

④

tidb-pd

tidb-tikv

tidb-tidb

cpu\_usag e

leader\_count

available\_size

uptime

duration\_99th

.....

规则手动配置，易于修改和扩展

⑤

pod\_trace: 统计自调用和外部调用耗时信息

abnormal\_status\_code: 异常code

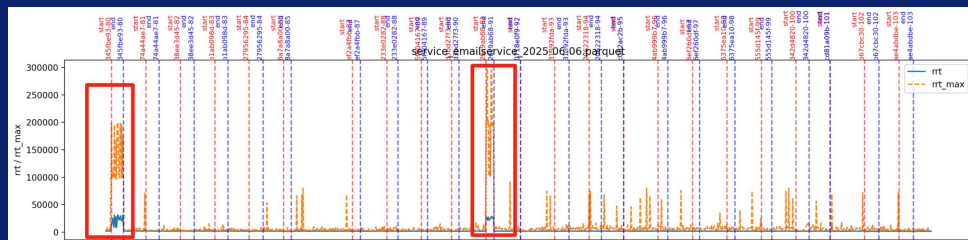
server\_call\_count: 每个服务的不同pod调用次数

```
'checkoutservice-0': {  
  'self_calls_top5': [{  
    'operationName': 'hipstershop.CheckoutService/PlaceOrder',  
    'duration': 221838,  
    'tags': {  
      'status_code': '0',  
      'status_message': '',  
      'span_kind': 'server'  
    }  
  }],  
  'process': {  
    'serviceName': 'checkoutservice',  
    'ip': '10.233.77.53',  
    'name': 'checkoutservice-0',  
    'node_name': 'aiops-k8s-04'  
  }  
},  
'span_start_dt': '2025-06-17T21:06:10.853000+00:00'  
}],  
'duration_avg_top5': 69032.4,  
'duration_avg_bottom5': 3997.6,  
'duration_avg': 10167.298245614010,  
'external_calls_avg_top5_stats': [{  
  'target_service': 'smallservice',  
  'duration_avg_top5': 82647.8,  
  'duration_avg_bottom5': 4263.8,  
  'duration_avg': 16028.793812940642
```



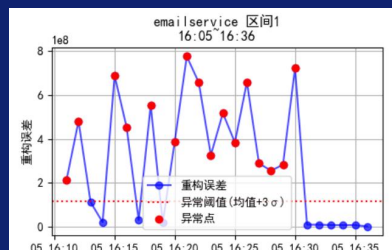
# 挑战二：如何有效准确的进行根因定位？

比赛并没有提供异常定位的任何方法说明 -> 分析数据，提出面向多源异构数据的融合过滤方法



①

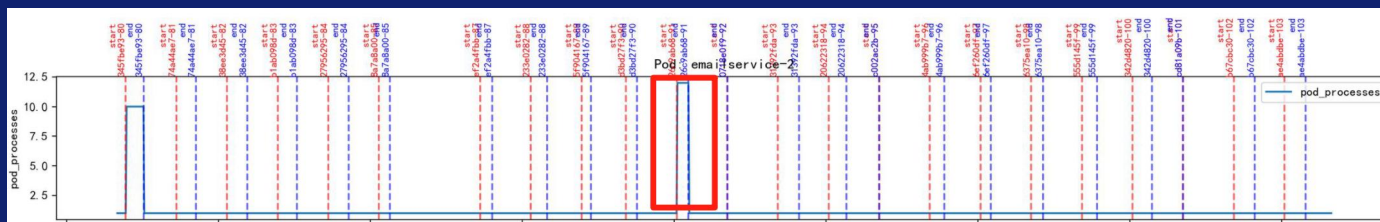
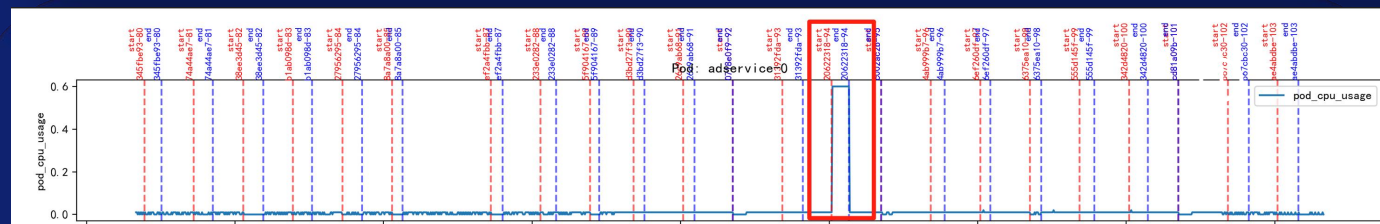
rrt / rrt\_max 明显波动，可能为异常。



```
# 日志过滤关键词
keywords = [
    "error", "exception", "failed", "fail", "failure", "abort", "panic", "critical",
    "fatal", "reject", "refused", "denied", "invalid", "unavailable", "timeout", "not found",
    "unreachable", "lost", "dropped", "disconnect", "corrupted", "mismatch", "conflict",
    "overload", "exhausted", "warning", "assertion", "rollback", "recoverable", "unrecoverable",
    "interrupted", "incomplete", "not allowed", "unauthorized", "forbidden", "unsupported",
    "unhandled", "unexpected", "deprecated", "missing", "blocked", "stopped", "halted", "broken",
    "crashed", "issue", "trouble", "degraded", "skipped", "overflow", "underflow",
    "hang", "stall", "block", "pause", "deadlock", "jam", "stop", "GC",
    "invalidated", "expired", "inconsistent", "illegal", "unknown", "shutdown",
    "retry", "reset", "revoked", "suppressed", "ignored", "unresponsive"
]
```

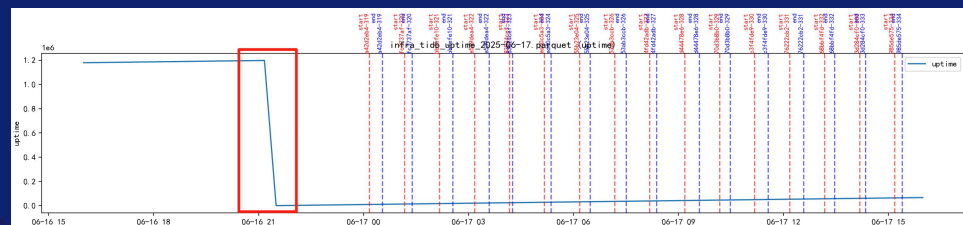
③

需要关注的异常关键词



②

部分 infra存在明显波动的指标，可能为异常。



④

```
'checkoutservice-0': {
  'self_calls_top1': [
    {
      'operationName': 'hipstershop.CheckoutService/PlaceOrder',
      'duration': 221038,
      'tags': {
        'status.code': '0',
        'status.message': '',
        'span.kind': 'server'
      }
    }
  ],
  'process': {
    'serviceName': 'checkoutservice',
    'ip': '10.233.77.53',
    'name': 'checkoutservice-0',
    'node_name': 'aiops-k8s-04'
  },
  'span_start_dt': '2025-06-17T21:06:16.853000+00:00'
},
'duration_avg_top5': 69632.4,
'duration_avg_bottom5': 3997.6,
'duration_avg': 10167.298245614034,
'external_calls_avg_top5_stats': [
  {
    'target_service': 'emailservice',
    'duration_avg_top5': 82647.8,
    'duration_avg_bottom5': 4763.0,
    'duration_avg': 16628.733812949642
  }
]
```

⑤

自调用和外部调用耗时

```
'frontend-0': {
  'frontend': [
    {
      'operationName': 'hipstershop.Frontend/Recv.',
      'duration': 25083,
      'tags': {
        'status.code': '13',
        'status.message': 'HTTP status code: 500',
        'span.kind': 'server',
        'error': 'true'
      }
    }
  ],
  'process': {
    'serviceName': 'frontend',
    'ip': '10.233.77.21',
    'name': 'frontend-0',
    'node_name': 'aiops-k8s-04'
  },
  'span_start_dt': '2025-06-17T21:11:50.893000+00:00'
}
```

异常code

```
'checkoutservice': {
  'frontend': {
    'frontend-2': 87,
    'frontend-0': 15,
    'frontend-1': 43
  }
}
```

来自外部调用次数

# 挑战二：如何有效准确的进行根因定位？

让模型清楚如何基于输入的数据进行根因定位？ -> 知识库：Prompt / RAG

尝试设计了多种prompt，最后采用了以下结构

” 运维智能指令五段式”：

- a) 角色 (Role): 场景中特定角色或功能。
- b) 相关信息 (Relevant info) : 所有必要和相关的细节。
- c) 行动 (Action): 明确请求或目标。
- d) 结果 (Result): 希望实现什么。
- e) 示例 (Example): 一个例子帮模型理解请求。

为大模型注入领域知识、明确任务约束  
并提供示例示范，更高效精准地驱动大  
模型完成微服务故障根因定位。

a

You are an operations expert specializing in identifying faulty components and root causes in microservices systems through data analysis.

b

System Architecture & Deployment:

The frontend directly depends on multiple backend microservices, including adservice, cartservice, checkoutservice, currencyservice, emailservice, paymentsservice, productcatalogservice, recommendationservice, and shippingservice. ....

- Deployment: 10 core microservices, each with 3 Pods, dynamically scheduled across 8 VMs (nodes). ....

c

1. Only output the single most likely faulty component name not ip (Pod, Service, or Node).
2. If all three pods of a service are abnormal, consider the service as faulty.....

d

Output format (strictly use the format below, with no extra content):

First,the reason should briefly summarize the reasons, such as network error, CPU stress, etc. Then, state which metrics indicate the abnormality, keeping it as concise as possible.Keep it under 20 words.....

e

For example: component: checkoutservice reason: disk IO overload  
reasoning\_trace:[ {"step": 1,"action": "LoadMetrics(checkoutservice)","observation":  
"disk\_read\_latency spike ".....}]

实验表明：相比无组织的提示词，得分有至少 5 分的提升。

# 挑战三：如何保证实际应用可行性？

## 1. 模块化设计 — 易修改、扩展和插拔

### Infra

metric-infra-pod

metric-infra-node

cpu\_usage\_rate

filesystem\_usage\_rate

pod\_processes

...

```
metric_funcs = {  
    'pod_cpu_usage': pod_cpu_usage,  
    'pod_processes': pod_processes,  
    # 'pod_network_transmit_bytes': pod_network_transmit_bytes,  
    # 'pod_network_receive_bytes': pod_network_receive_bytes,  
}
```

新增/删除检测指标只需要修改map，方便扩展

tidb-pd

tidb-tikv

tidb-tidb

uptime

cpu\_usage

.....

duration\_99th

log

"error", "exception", "failed", "fail", "failure", "abort", "panic", "critical", "overflow", "underflow",  
"hang", "stall", "block", "pause", "deadlock", "jam", "stop", "GC", "retry", "reset", "revoked", "suppressed", "ignored", "unresponsive"

新增/删除关键词

trace

pod\_trace: 统计自调用和外部调用耗时信息

abnormal\_status\_code: 异常code

server\_call\_count: 每个服务的不同pod调用次数

新增/删除字段并实现相应方法



# 挑战三：如何保证实际应用可行性？

## 2. 易扩展为Multi-Agent – 分层式设计（方案1）



RRT  
Agent

RRT

metric-apm-pod

metric-apm-service

rrt

rrt\_max

LSTM预测误差 均值+3std

error

time out

>0 即可能为异常

Function Call / MCP



Infra  
Agent

Infra

metric-infra-pod

metric-infra-node

cpu\_usage\_rate

memory\_usage\_rate

filesystem\_usage\_rate

pod\_processes

...

Function Call / MCP



Log  
Agent

log

Function  
Call /  
MCP

"error","exception","failed","fail","failure","abort","panic","critical",  
"overflow","underflow",  
"hang","stall","block","pause","deadlock","jam","stop","GC",  
"retry","reset","revoked","suppressed","ignored","unresponsive"



Tidb  
Agent

tidb-pd

tidb-tikv

tidb-tidb

cpu\_usag

e

leader\_count

available\_size

uptime

duration\_99th

.....

Function Call / MCP



Trace  
Agent

trace

pod\_trace: 统计自调用和外部调用耗时信息

abnormal\_status\_code: 异常code

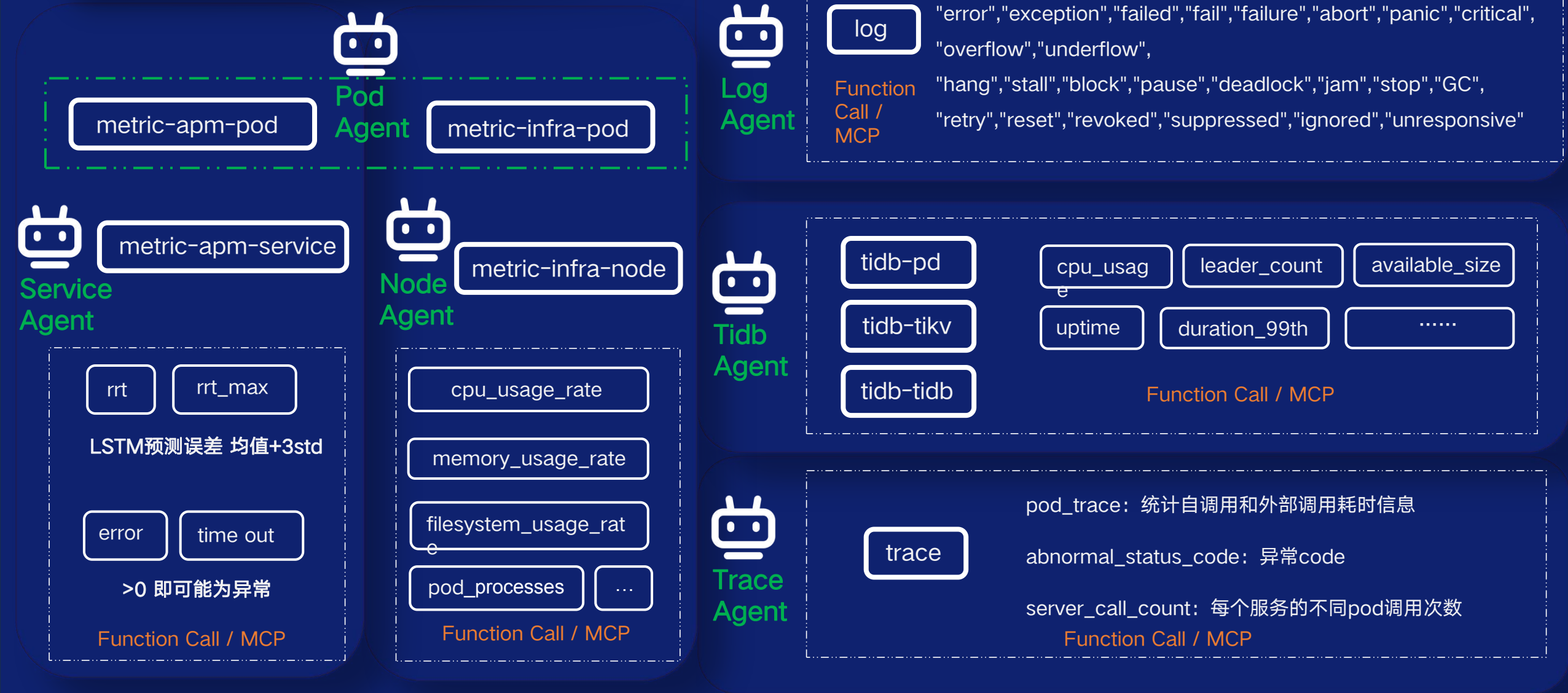
server\_call\_count: 每个服务的不同pod调用次数

Function Call / MCP



# 挑战三：如何保证实际应用可行性？

## 2. 易扩展为Multi-Agent – 分层式设计（方案2）



# 挑战三：如何保证实际应用可行性？

## 3.减少Token消耗 - 多源数据的Token高效约束采样方法

### RRT

metric-apm-pod

metric-apm-service

rrt

rrt\_max

LSTM预测误差 均值+3std

按误差最大值排序**取前半**

service/pod max\_error

error\_top3\_mean threshold

max\_error\_detail

anomaly\_count

error

time out

>0 **限制取10条**

### Infra

metric-infra-pod

metric-infra-node

cpu\_usage\_rate

memory\_usage\_rate

filesystem\_usage\_rate

pod\_processes

...

只保留**部分必要字段**

如: ["time", "instance",  
"kpi\_key", "object\_type",  
"pod", "pod\_cpu\_usage"]

log

每个关键词

**限制取10条**

"error","exception","failed","fail","failure","abort","panic","critical",  
"overflow","underflow",  
"hang","stall","block","pause","deadlock","jam","stop","GC",  
"retry","reset","revoked","suppressed","ignored","unresponsive"

tidb-pd

tidb-tikv

tidb-tidb

cpu\_usage

leader\_count

available\_size

uptime

duration\_99th

.....

只保留**部分必要字段**

trace

字段转换,

**提取关键信息**

pod\_trace: 统计自调用和外部调用耗时信息

abnormal\_status\_code: 异常code

server\_call\_count: 每个服务的不同pod调用次数





## 第三章

# 总结

目标：构建一个高效利用数据、准确进行根因定位、高工程实用性的大模型根因定位系统

## 挑战三：如何保证实际应用可行性？

Docker容器化部署，支持多进程并发与异步队列，支持高并发场景下的高效处理

多维度数据的模块化可扩展检测方法，支持分钟级扩展新检测维度

分层式协同数据处理，方便扩展为Multi-Agent架构

多源数据的Token高效约束采样方法，显著降低大模型处理成本。

## 挑战一：如何高效利用海量数据？

多模态数据差异化异常检测与精准输入

数据交叉验证实现全栈监控

## 挑战二：如何有效准确的进行根因定位？

提出面向多源异构数据的融合过滤方法，高效检测异常数据。

提出“运维智能指令五段式”prompt设计范式，高效精准地驱动大模型完成微服务故障根因定位

## 1. 多模态数据融合

整合时延、基础指标、日志、Trace、TiDB组件状态 5 类数据，通过**交叉验证实现全栈监控**，显著提升数据利用率和异常检测准确性。

## 2. 多样化异常检测算法

设计多样化异常检测算法体系，包括RRT(LSTM时序预测与3σ统计结合)、资源(配置规则)、log(关键词匹配)、trace(数据聚合分析)等。

## 3. LLM驱动的因果推理

设计LLM驱动的因果推理框架，通过DeepSeek-R1等大模型与专家规则库协同，自动生成“根因锚点—关联影响—交叉验证”的**可解释诊断报告，突破单维度分析局限**。提出“**运维智能指令五段式**”prompt设计范式，高效精准地驱动大模型完成微服务故障根因定位。

## 4. 工程高可落地性

工程化落地智能运维系统，采用Docker容器化部署，支持多进程并发与异步队列，实现高并发场景下的实时处理；模块化设计支持**分钟级扩展新检测维度**，同时方便**扩展为Multi-Agent架构**；多源数据的Token**高效约束采样**方法，显著降低大模型处理成本。

## 5. 精准故障定位能力

实现微服务环境下精准的多级故障定位能力，故障识别覆盖核心业务服务及TiDB/Redis基础设施，可以**准确识别**服务级/Pod级/节点级故障(如emailservice CPU异常、adserviceGC错误等)，并**提供完整证据链**。

OpenAIOps AIOPS | 2025 CCF国际AIOps挑战赛  
2025 CCF International AIOps Challenge

# THANKS

主办单位：中国计算机学会（CCF）

承办单位：中国计算机学会互联网专委会、中国科学院计算机网络信息中心、中国移动研究院、清华大学

协办单位：华为2012实验室、阿里云、中兴通讯、中国移动九天团队、南开大学、西安电子科技大学、清华大学计算机科学与技术系、神州灵云



北京邮电大学



网络与交换技术全国重点实验室  
State Key Laboratory of Networking and Switching Technology

# 附录一 异常检测方法-Service

## 基于LSTM神经网络和3 $\sigma$ 统计方法的服务级异常检测方法：

### LSTM神经网络

- 时序建模：使用5个时间窗口的历史数据预测下一个时间点
- 非线性激活：Softplus激活函数确保响应时间预测值为正
- 轻量级设计：64维隐藏层，平衡精度与性能

### 3 $\sigma$ 异常检测算法

优选最小10个误差进行统计

### 算法优势

自适应阈值：基于预测误差分布动态调整

抗噪能力：使用最小误差集合避免异常值干扰

统计稳定性：3 $\sigma$ 准则确保大部分正常数据不被误判

计算每个服务的max\_error总和 -> 降序排列，选择前一半的服务

## 基于规则的异常检测方法：

- 非零检测：error或time out任一错误字段 > 0 即认为异常
- 字段筛选：只报告有异常值的字段



# 附录一 异常检测方法-Pod

沿用Service的处理方法，针对每个pod进行异常检测。

服务级模型训练 → Pod级检测复用

优势：

数据充分：服务级聚合数据更丰富，模型更稳定

一致性：同一服务的Pod使用统一的性能基准

效率：避免为每个Pod维护独立模型

理论依据：

同一服务的不同Pod实例应该有相似的性能模式

Pod间的差异主要体现在负载分布而非性能特征

多指标Pod异常检测：

对多个Pod资源指标（如pod\_cpu\_usage、pod\_processes等）进行异常检测。

检测逻辑灵活，可根据不同Pod类型设置不同阈值（如redis-cart进程数大于2为异常，其它为大于1）。





检测Kubernetes集群中各个节点的基础设施指标异常。通过统计学方法和双阈值检测，识别节点在CPU、内存、磁盘、网络等关键指标上的异常行为。

- **多维度监控:** 涵盖CPU使用率、内存使用率、文件系统使用率等关键基础设施指标
- **智能异常检测:** 基于阈值检测和差值分析的双重异常识别机制
- **分组处理:** 支持按mountpoint等维度进行分组分析，提高检测精度
- **时序分析:** 基于时间序列的逐项差值异常检测，识别突变和波动
- **灵活配置:** 支持多种检测策略和阈值配置

## 双阈值检测:

- 绝对阈值（如  $\text{cpu} > 40\%$ ）瞬时越界告警；
- 差分阈值（如内存增长率  $> 20\%$ ）捕捉“斜坡式”泄漏，弥补单点阈值对缓慢恶化不敏感缺陷。

## 轻量级无监督:

纯统计计算，无需标签、无需训练，新增指标仅几行配置即可接入，实现“热插拔”式扩容。

# 附录一 异常检测方法-TiDB

通过监控 TiDB 集群的三个核心组件（PD、TiDB、TiKV）的关键指标，实现全方位的数据库健康状态监控。

- 模块化设计: 各组件检测逻辑独立，便于维护和扩展
- 统一接口: 提供一致的调用接口和返回格式
- 容错机制: 单个组件检测失败不影响其他组件

## PD:

**多维度监控:** 涵盖存储节点状态、存储空间、性能指标及集群拓扑等多方面核心指标，反映系统整体健康状况。

**智能异常检测:** 结合阈值判断（如节点数量、空间占用、CPU使用率等）与变化趋势分析（如存储大小突降、CPU持续上升）双重机制，实现关键异常的智能识别。

## TiDB Server:

**多维度监控:** 覆盖服务可用性、查询性能以及系统资源等核心指标，全面反映系统健康状态。

**智能异常检测:** 采用阈值判断和趋势分析相结合的方式，精准识别如服务离线、连接数异常、查询延迟升高、CPU负载上升、服务重启等关键异常。

## TiKV:

**多维度监控:** 覆盖查询性能（QPS/gRPC QPS）、系统资源（CPU、内存、IO）及存储空间等关键指标，全方位反映系统运行状态。

**智能异常检测:** 结合阈值判断与波动趋势分析，精准识别如QPS剧烈波动、CPU过载、内存持续增长、IO负载过高、可用空间突变等典型异常。



# 附录一 异常检测方法-Trace

分析微服务间的调用链路，识别性能异常、状态码异常和服务间调用模式。通过追踪数据，提供深度的服务调用性能洞察和异常检测能力。

## 1. 全方位、多维度监控

**覆盖性能、错误、依赖三大核心维度：**监控服务的自调用与外部调用性能，调用过程中的异常状态码，以及服务之间的调用关系，形成对微服务系统的全景式画像。

**Pod级别精细化监控：**分析结果细化至Pod实例维度，支持对单个Pod的性能瓶颈、异常服务调用等问题进行精准定位。

## 2. 性能分析与瓶颈识别

**自调用与外部调用分离分析：**区分服务自身内部方法调用与对外部服务的调用，分别进行性能排序（Top1/Top5/Bottom5），有效识别最耗时的自调用和影响整体性能的外部依赖。

**多统计指标体系：**提供如Top1最耗时调用、Top5/Bottom5平均耗时、总体平均耗时等多维度统计指标，帮助定位慢查询、突发延迟等性能异常点。

**多目标服务分组与排序：**外部调用分析支持按目标服务分组，进一步通过性能指标排序找出最慢的外部依赖

## 3. 异常状态码捕捉

**细粒度异常检测：**识别非零状态码的服务调用，且支持按目标服务分组。

**异常信息丰富：**异常保留详细的上下文信息（如操作名、耗时、状态码、时间戳等），为后续故障分析和自动化告警提供依据。

## 4. 服务依赖关系全景还原

**动态服务依赖图谱构建：**统计服务间、Pod间的调用次数，构建微服务依赖链路，辅助识别流量异常、调用热点等系统级问题。



# 附录一 异常检测方法-Log

通过关键词匹配的方式，从大量的 Pod 日志中快速定位包含错误、异常、警告等问题信息的日志条目。

简单高效：基于字符串匹配，性能优异

准确性高：精心设计的关键词库，误报率低

可扩展：关键词库可以根据实际情况调整

## 四层关键词库：

错误类 / 性能类 / 系统状态 / 数据一致性 87 词

## 时间边界扩展：

自动扩展查询时间段（默认前后各5分钟），避免因时钟偏差或日志延迟造成的异常遗漏。

## Pod级别采样与分批处理：

每个Pod最多采样20条异常日志，并采用分文件、分批处理方式，保障系统内存占用可控且结果具备代表性与可重现性。



## Role:

You are an operations expert specializing in identifying faulty components and root causes in microservices systems through data analysis.

## Relevant info:

### System Architecture & Deployment:

User interacts with the system exclusively through the frontend service. The frontend acts as the single entry point for all system traffic and is the only service directly exposed to users.

The frontend directly depends on multiple backend microservices, including adservice, cartservice, checkoutservice, currencyservice, emailservice, paymentservice, productcatalogservice, recommendationservice, and shippingservice.

The dependencies among backend services are as follows:

Recommendationservice relies on productcatalogservice to obtain product details for generating recommendations.

Checkoutservice, during the checkout process, relies on shippingservice, paymentservice, currencyservice, and emailservice for shipping, payment processing, currency conversion, and sending confirmation emails.

Cartservice uses redis-cart as its caching or storage backend.

Productcatalogservice and adservice depends on the underlying tidb database for storing and retrieving product information. tidb consists of three core components: tidb-tidb, tidb-tikv, and tidb-pd.

The TiDB components together form a distributed database cluster supporting the data requirements of productcatalogservice and adservice.

All service calls are unidirectional: the frontend initiates all requests, and some backend services may in turn call their downstream dependencies.

Except for cartservice, which depends on redis-cart, and productcatalogservice, which depends on tidb, the other services are independent and do not rely on external services.

- Deployment: 10 core microservices, each with 3 Pods, dynamically scheduled across 8 VMs (nodes).

- Fault injection has three types:

- Service-level: All 3 Pods of a service fail simultaneously;
- Pod-level: A single Pod of a service fails;
- Node-level: All Pods on a VM fail.

## Action:

### Rules:

1. Only output the single most likely faulty component name not ip (Pod, Service, or Node).
2. If all three Pods of a service are abnormal or multiple nodes are abnormal, consider the service as faulty.
3. If only a single Pod is abnormal, consider the Pod as faulty. It is necessary to carefully distinguish this from the node failure described below.
4. If a node fails and affects multiple services (services or Pods), consider the node as faulty. If multiple nodes fail, it should be considered a service fault, and the corresponding service should be identified.
5. Before identifying a node as faulty, you also need to check whether any pods are abnormal, because pod abnormalities may also cause corresponding node anomalies. In such cases, the root cause is not the node, but the pod.
6. The following scenarios require special attention. You should pay particular attention to input Pods with suspicious metrics, potential faulty nodes and their suspicious metrics, and potential faulty TiDB data. If the data matches the rules below, it is highly likely to be the root cause of the failure—please pay extra attention! Give priority to these situations before considering network or other types of anomalies.

### Pod:

pod\_processes: For redis-cart pods, a process count greater than 2 is abnormal; for other pods, a process count greater than 1 is abnormal and may indicate abnormal CPU stress.

pod\_cpu\_usage: CPU usage greater than 0.2 may indicate abnormal CPU stress.

### Node:

node\_cpu\_usage\_rate: Greater than 0.4 or an increase exceeding 0.2 is abnormal and may indicate abnormal CPU stress.

node\_filesystem\_usage\_rate: An increase exceeding 0.2 is considered abnormal.

node\_memory\_usage\_rate: An increase exceeding 0.2 is considered abnormal and may indicate abnormal memory stress.



tidb-pd:

cpu\_usage: An increase exceeding 0.2 within the time window is considered abnormal and may indicate abnormal CPU stress.

store\_size: A decrease between consecutive samples is considered abnormal.

tidb-tidb:

connection\_count: A value less than 1 is considered abnormal.

duration\_99th: A value greater than 1 is considered abnormal.

duration\_avg: A value greater than 0.02 is considered abnormal.

uptime: A decrease between consecutive samples is considered abnormal, indicating tidb-tidb was killed.

cpu\_usage: An increase exceeding 0.1 within the time window is considered abnormal and may indicate abnormal CPU stress.

tidb-tikv:

cpu\_usage: Abnormal only if value >0.3 and increase >0.2; may indicate CPU stress.

available\_size: An increase exceeding 1e10 is considered abnormal.

store\_size: A decrease between consecutive samples is considered abnormal.

7. For network-related issues, both the caller and callee should be specified, but only output one of the involved service names; do not output pod names.

8. If there is an issue with the TiDB database, only return the corresponding TiDB component (tidb-tidb, tidb-tikv, or tidb-pd), not the Pod.

9. First, check for anomalies in rrt, rrt\_max, trace duration, and logs for each input service, and also check the metrics of each input pod, node, TiDB, and redis-cart. If any obvious anomalies are found, mark the corresponding component as faulty and then perform further verification and analysis.

10. For network-related metrics (such as QPS, network\_receive\_packets), confirm whether values have dropped significantly—a value of zero alone does not prove failure, as upstream network issues may simply block requests; comprehensive analysis is required.

11. There is no dependency relationship between redis-cart and TiDB; they are two independent services.

12. Data for certain services and some service pods may be missing during specific time periods, so a comprehensive approach to data utilization is required.

13. Root cause identification and analysis must be supported by metrics data; rough or speculative judgments are not acceptable.

14. If abnormalities are found in rrt, rrt\_max, client\_error, etc., but no anomalies are found in node or pod metrics such as cpu or memory\_usage\_rate, nor in logs or trace data, then it should be identified as a network error. Clearly specify both the caller and callee services, and only output service names, not pod names.

15. For components such as redis-cart, currencyservice, tidb-tidb, tidb-pd, and tidb-tikv, root cause should only be identified if you are very certain, as the metrics of these components often fluctuate and do not necessarily indicate actual faults. However, this does not mean they can never be the root cause, so do not exclude them completely.

16. Special attention should be paid to tidb-tidb, tidb-pd, tidb-tikv, and redis-cart: decreases in cpu\_usage and memory\_usage\_rate are generally not considered anomalies. However, a significant increase is usually considered abnormal and should be judged in combination with the time sequence.

17. For anomalies in rrt, rrt\_max, client\_error, etc., first identify the immediate caller and callee (for example, by analyzing traces), rather than blindly attributing the fault to redis-cart.

18. Keywords such as "error" or "500" observed in logs may indicate that the pod has encountered issues for various reasons (e.g., being killed), and are not necessarily network errors. However, it is crucial to carefully determine whether the issue is due to a network problem between two services or an error within a single pod, as distinguishing between these two scenarios is very important.

19. For 'external\_calls\_avg\_top5\_stats', you need to compare the data of all target services horizontally. Only when the duration\_avg\_top5 of a certain target\_service shows a clear anomaly should you further check the CPU, memory, and other metrics provided by the pod and node, as well as logs, to determine whether there is a network issue between the two services. You should not easily draw conclusions based solely on rrt, rrt\_max, or similar data.

20. The root cause analysis must be supported by data and should be corroborated with the reasoning trace; arbitrary speculation is not allowed.



21. It is important to clearly distinguish whether the issue is at the pod level or the service level. Network errors between two services should be considered as service-level issues. If only a certain pod shows abnormal metrics, it is a pod-level issue. You need to observe all pods under the service—if all of them have problems, then it is considered a service-level issue.
22. Special attention must be paid to keywords such as "gc" in logs; if such keywords are observed, it is highly likely that a GC-related fault has been injected, i.e., the fault type is gc error. Similarly, pay attention to error keywords like "error" or "failed" in logs.
23. Judgments should only be made after comprehensively considering all the above factors.

## Result:

Output format (strictly use the format below, with no extra content):

In the reason, clearly specify which metrics or data sources support your conclusion, such as rrt, rrt\_max, log error, cpu\_usage, etc.

First, the reason should briefly summarize the reasons, such as network error, CPU stress, etc. Then, state which metrics indicate the abnormality, keeping it as concise as possible. Keep it under 20 words.

Complete the reasoning\_trace in as few steps as possible, but ensure the steps are coherent. component:XX reason:xxx reasoning\_trace:xxx

## Example:

For example:

component: checkoutservice

reason: disk IO overload

reasoning\_trace:[

```
{{
  "step": 1,
  "action": "LoadMetrics(checkoutservice)",
  "observation": "disk_read_latency spike"
}},
{{
  "step": 2,
  "action": "TraceAnalysis('frontend -> checkoutservice')",
  "observation": "checkoutservice self-loop spans"
}},
```

```
{{
  "step": 3,
  "action": "LogSearch(checkoutservice)",
  "observation": "IOError in 3 logs"
}}
]
```

## Input Data:

- Potential faulty service: {abnormal\_service\_map}
- Potential faulty pods: {abnormal\_pod\_map}
- Pods with suspicious metrics: {pod\_metric\_map}
- Suspect logs: {log}
- Call chain traces: {pod\_trace\_result}
- Abnormal status codes during calls: {abnormal\_status\_code}
- Potential faulty node and suspicious metrics: {node\_metric\_map}
- Count of invocations by external services: {server\_call\_count}
- Potential faulty TiDB data: {abnormal\_tidb}

Carefully analyze all the provided information and system architecture. Output only ONE most likely faulty component and its underlying root cause, strictly following the requirements above. Try your best, please!