基于事件存储引擎的可观测系统的设计与实现

陈红茜[1,2]，邱小彬[1]，屈阳[1]，曹磊[1]，王居正[1]，陈昕[1,3]
（1. 中国农业大学信息化办公室（网络技术中心），北京 100083；2. 农业农村部设施农业工程重点实验室，北京 100083；
3. 中国农业大学信息与电气工程学院，北京 100083）

摘　要：为了解决近 10 年软件架构剧变、新技术普及引起的软件系统复杂性增加，从而使软件 bug、系统故障排查困难陡增等问题，针对起源于传统单体服务的监控系统，提出了一种更具灵活性、效率更高的监控系统的设计、实现方法。将传统的监控数据源进行抽象，统一为事件模型，并设计相应的存储引擎，提供统一的查询、写入 API，最后基于该事件存储引擎构建了可观测系统，提供更加丰富、强大的查询、分析能力。最终的应用效果表明，相对于传统的监控系统，所构建的可观测系统在故障排查、问题分析等方面的效率得到了大幅提升。

关键词：可观测系统；事件存储引擎；监控系统；分布式
中图分类号：TP315
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2024256

Design and implementation of observability system based on event storage engine

CHEN Hongxi[1,2], QIU Xiaobin[1], QU Yang[1], CAO Lei[1], WANG Juzheng[1], CHEN Xin[1,3]
1. Network Technology Center, China Agricultural, Beijing 100083, China
2. China Agricultural University Key Laboratory of Agricultural Engineering in Structure and Environment of MOARA, Beijing 100083, China1.
3. College of Information and Electrical Engineering, China Agricultural University, Beijing 100083, China.

Abstract: To solve the challenges posed by the significant increase in software system complexity due to the drastic changes in software architecture and the widespread adoption of new technologies over the past decade—challenges that have led to a surge in software bugs and difficulties in system failure troubleshooting, a design and implementation method for a more flexible and efficient monitoring system tailored to traditional monolithic service-based monitoring systems was proposed. The method abstracted traditional monitoring data sources into a unified event model and designed a corresponding storage engine that offered a unified query and write API. Based on the event storage engine, an observability system was constructed, providing richer and more powerful querying and analysis capabilities. The final application results demonstrate that, compared to traditional monitoring systems, the observability system developed in this paper significantly enhances efficiency in troubleshooting and problem analysis.

Keywords: observability system, event storage engine, monitor system, distributed

0　引言

近 10 年软件架构发生了巨大的变化，随着云原生、微服务、虚拟化、Service Mesh、自动扩缩容等新技术的普及，现代软件系统变得越来越复杂。典型地，一个用户请求可能会经过数十次网络请求，导致问题的排查相对于传统单体系统难度增加了很多[1-3]。所以，单体服务时代流行的监控系统在目前的复杂软件架构下存在问题排查效率低下

收稿日期：2024–08–15
通信作者：陈昕，chxin@cau.edu.cn
基金项目：科技创新 2030-“新一代人工智能”重大基金资助项目（No.2021ZD0113800）
Foundation Item：The National Key Research and Development Program of China (No.2021ZD0113800)

的问题。传统的监控系统（如日志搜索、Metric等）诞生于单体系统时代，已经发展数十年，并已经成为软件从业人员心中的最佳实践。然而，随着软件技术的发展，软件架构的演进，软件系统复杂度迅速增加。传统软件系统与现代软件系统比较如表1所示。

在引入了微服务、多种存储、Service Mesh、容器化、Lambda Function等技术后，系统变得越来越复杂，所以和单体应用相比，现在最难的事情已经从代码的debug变成了找到问题出现的根因。例如，找到系统的性能瓶颈越来越困难，某个组件变慢了，可能导致调用链上游所有的服务都变慢了。更有甚者，问题可能出现在不受控制的系统里。所以，当系统出现问题时，需要借助一些高效率的工具来解决。

为了解决软件复杂性提高带来的问题排查的效率问题，业界提出了可观测性这一概念：在软件工程中，更具体地说，在分布式系统中，可观测性是收集有关程序执行、模块内部状态和组件之间通信数据的能力。通过对这些数据的分析增强工程师对于系统状态的理解，从而提升问题排查效率。通常，为了提高可观测性，系统中广泛使用日志（Log）、分布式追踪（Trace）和指标（Metric）等来收集数据，并通过相关的工具对这些数据进行分析。其中，Log、Trace和Metric又被称为“可观测性的三大支柱”[4]。

Log记录了系统中发生事件[5]，并且具备极高的价值，开发及运维人员通常会使用日志对系统中发生的问题进行定位和分析[6-10]。由程序输出的非结构化日志采用叙述结构，主要方便人类阅读，但是不利于机器处理[11]。通常，日志会有比较大的噪声，只有在开发人员已经明确出现的问题时，日志才能更好地发挥作用，这时可以通过搜索某个关键字来定位问题（所以要求对系统熟悉）。之前日志都是存储在本地磁盘，可以方便地去本地磁盘查看日志，现在一般都会将日志发送到后端进行存储，例如开源的ELK或者商用的Splunk。

Metric表示某些系统状态在某一时刻的测量值，是一段时间内的预聚合结果，上报到后端的数据就是最小粒度，无法进一步分析，由此可能会掩盖真正的问题[12]。常见的Metric系统包括Zabbix、OpenTSDB、Prometheus等。

Trace一般指代分布式追踪，用于对分布式系统进行故障诊断、性能调优、系统理解以及资源审计[13-14]。在分布式系统下，通过一系列相互关联的离散事件（被称为Span，即跨度）来追踪单个用户请求。通过记录请求在分布式系统中每个服务下的Span，从而形成一个服务调用链。可以显示为了处理这个请求，这些相互操作的服务之间的因果、事件关系，从而能够排查性能瓶颈。常见的开源实现包括Skywalking、Jaeger等。

通常，以上3种数据需要用到不同的存储系统，本文提出通过事件存储引擎统一存储以上3种数据，同时基于事件存储引擎设计并实现可观测系统，提供多种高级智能的分析手段，提升问题排查效率。

1 事件模型

Log、Trace、Metric是传统监控系统经常用到的数据类型，三者的数据粒度越来越粗，通常，一个Metric是对一段时间内方法调用的聚合统计，Trace中的Span可以对应一次方法调用，而在一次方法调用中可以有多个Log的输出。在进行数据

表1 传统软件系统与现代软件系统比较

指标	传统软件系统	现代软件系统
服务数量	单体应用	多个（微服务）、分布式
存储系统	一个数据库系统	多个（种）持久化存储系统
部署方式	静态	动态，弹性伸缩
可控制	完全可控	不一定能够直接控制
面向人群	运维工程师	开发工程师
稳定性重点	服务运行时间和故障预防	主动检测代码变更，尽早发现问题；可以容忍降级
复杂度	低	高

分析时，会首先通过选择一段时间范围查看 Metric，判断是否有异常，如果有异常再深入 Trace 确定异常发生在哪个服务的哪个方法，最终可以通过该方法的 Log 确定根因。上述是 3 种观测数据通常的使用方法，本文基于三者的共性抽象出事件模型，从而可以方便、统一地存储这 3 种数据，降低系统维护的复杂度。

1.1　事件模型设计

Log、Trace、Metric 的典型示例如下。

1) Log

一条 Log 主要有时间戳以及表示日志内容的 message 字段，其他所有的字段可以认为是 Log 的一些维度。基于这些维度可以进行搜索和聚合。

```
{
“timestampNs”: 1722953210000000000，
“message”：“user test has logged in”，
“level”：“INFO”，
“host”：“hostname-1”，
“service”：“auth-service”，
“durationMs”: 10,
…
}
```

2) Trace

Trace 主要由一系列的 Span 组成，每个 Span 对应一次调用，其中，SpanId 标识一个 Span；traceId 表示 Span 属于哪 Trace 以及开始结束时间。其他的字段可以认为是维度或度量，用于搜索和聚合。

```
{
“spanId”：“yb7hagxatnj6s2udw3k-0”，
“traceId”：“ia7hagxatnj6t2udw3k”，
“startTime”: 1722953210000000000,
“endTime”: 1722953866000000000,
“duration”: 297,
“name”：“search-student”，
“status”：“ok”，
…
}
```

3) Metric

Metric 与 Log 类似，会有一个时间戳、指标值（node_meme_total_bytes）以及其他用于筛选的维度。

```
{
“timestamp”: 1722953210000
“node_mem_total_bytes”: 100000000,
“instance”：“192.168.1.10”，
“env”：“online”
“service”：“auth-service”，
…
}
```

通过以上 3 种数据的示例，可以发现三者的共性：1) 有标识时间的时间戳字段；2) 有用于筛选、分组聚合的维度信息；3) 有用于聚合的度量信息。基于这些共性，可以设计出表示三者的统一的 Event 事件模型如下。

一个 Event 包含系统字段：时间戳和事件类型，以及其他自定义字段。自定义字段可以是维度或度量，用于筛选、聚合等。

```
{
// 系统字段
“timestampNs”: 1722953210000000000，
“type”：“log”，
// 自定义字段
“message”：“user test has logged in”，
“level”：“INFO”，
“host”：“hostname-1”，
“service”：“auth-service”，
“durationMs”: 10,
……
}
```

通过事件模型可以很好地存储 Log、Trace 和 Metric 这 3 种数据，大幅简化数据的存储以及相应系统的运维。

1.2　数据类型

对于一个数据模型来说，其支持的数据类型是至关重要的，因为数据类型决定了模型的能力。针对 Log、Trace 和 Metric，本文提出的事件模型的数据类型如表 2 所示。

对于 Log、Trace 和 Metric，可以基于不同字段的功能选择相应的数据类型。比如，在 Log 中，message 字段表示日志的原文，是用户经常搜索的对象，所以需要提供分词、全文检索能力，可以设置为 Text 类型；相应地，对于耗时 duration 字段，通常需要计算平均值和分位值等，所以可以设置

为Long或Double类型。

表2 事件模型的数据类型

数据	类型	能力
Long	长整型	度量;搜索、聚合计算
Double	双精度浮点型	度量;搜索、聚合计算
String	字符串类型	维度;搜索、聚合分组
Text	文本类型	全文检索;搜索

1.3 查询接口

完成了事件模型和数据类型的设计后，接下来看一下模型需要提供的查询接口。对于Log、Trace、Metric在业务使用上需要支持：搜索、聚合分析，所以事件模型也必须提供同样的能力，具体的查询接口如下。

1) search

SearchResponse search(SearchRequest request)

search接口用于事件搜索，传入的搜索请求中会通过查询语言（QL, query language）定义搜索条件，例如：type = log AND service= “auth-service” AND duration > 10000 AND “log in, userId= 10000” AND timstampNS > 1720972800000000000。这个请求要检索的是：service是auth-service并且duration大于10000并且全文搜索 “log in, userId= 10000” 并且时间在1720972800000000000之后的所有log。

SearchResponse返回匹配的事件列表，这个列表按照事件时间排序，具体返回集合的大小由SearchRequest.limit参数定义。

2) searchAfter

SearchResponse searchAfter(SearchAfterRequest request)

searchAfter接口用于实现翻页，请求和响应结构体与search大体相同。不过，为了实现翻页，在响应体中每个匹配的事件都会有一个id属性，其中隐含了排序信息。用户翻页时，会取事件列表中最后一个事件的id属性并回传到服务端，表示搜索这个id之后的事件。

3) aggregate

AggregateResponse aggregate(AggregateRequest request)

aggregate接口表示进行聚合分析，在AggregateRequest中包含类似search的筛选条件。此外，还需要指定分组维度以及聚合算子。分组的维度只能是字符串类型，聚合算子要包括聚合操作类型以及聚合维度。聚合算子类型如表3所示。

表3 聚合算子类型

算子	功能
count	个数聚合
count distinct	去重后个数聚合
avg	平均值
sum	求和
max	求最大值
min	求最小值
percentiles	求分位值,比如P90、P95等

aggregate接口实现类似结构化查询语言（SQL, structural query language）的能力，比如：select count(distinct userId) from table group by service。通过分组、聚合后可以很容易对数据进行分析，发现离群值、异常值等。

事件模型通过支持以上查询接口，屏蔽了Log、Trace和Metric在实现细节上差异，可以很好地支持3种数据源在不同场景下的查询需求，为三者统一实现奠定了基础。

2 系统设计与实现

在本节，首先介绍系统实现的整体架构以及各个组件，然后详细讨论了事件存储引擎的实现细节，最后总结了系统的实现效果。

2.1 整体架构

通常，Log、Trace和Metric等观测数据具有数据量大、吞吐高的特征，同时需要尽量保证能够读取到最新数据。要满足上述需求，系统需要能够低时延、高吞吐地进行数据处理，整体系统采用分布式架构，同时通过流式数据处理降低时延性。系统整体架构如图1所示。

整体上，系统可以划分为两部分：写入侧和查询侧。顾名思义，写入侧主要是写入链路用于处理事件的高吞吐写入；查询侧用于处理用户的查询请求。

2.1.1 写入侧

写入侧的具体实现如下。

1) Agent

在写入侧，所有需要进行观测的节点上都部署

Agent 进程，用于对 Log、Trace、Metric 等观测数据进行采集。Agent 以守护进程的方式部署，为了提升吞吐同时避免占用过多的资源影响宿主进程，Agent 本地会进行缓存并以 batch 的形式将数据压缩并上传给后端。

2) 负载均衡

负载均衡（LB, load balancer） 是流量接入层，用于对请求进行路由、转发，从而使后端各个组件的流量是均衡的。由于 Agent 以 HTTPS 协议向后端发送数据，因此通过 LB 进行探活、负载均衡、防攻击等，从而实现后端高可用。

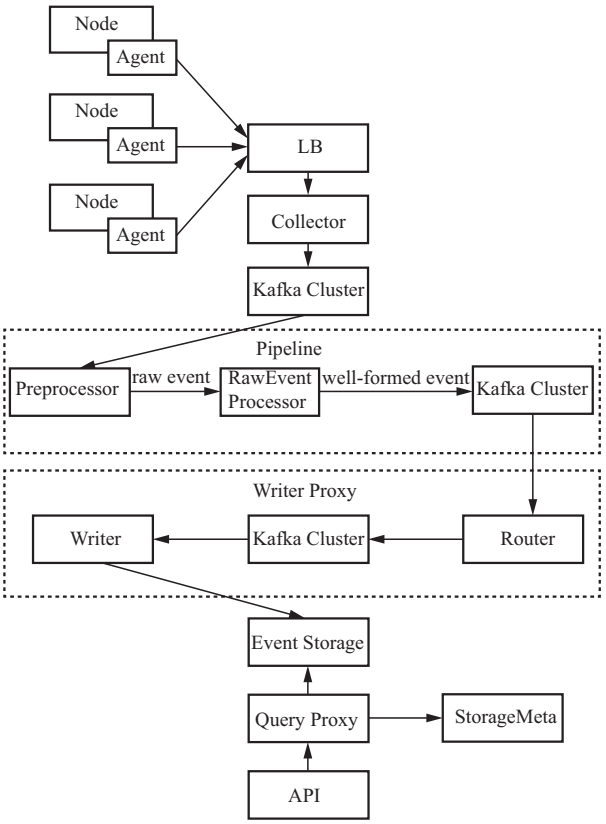


图1　系统整体架构

3) Collector

LB 会将请求转发给 Collector。Collector 将接收到的消息进行解压缩并还原成一个个的事件，然后发布到 Kafka 集群上。

4) Kafka 集群

Kafka 集群主要用于不同组件的异步通信，实现持久化、流量削峰等能力。

5) Pipeline

处理流水线 Pipeline 对事件进行处理、转换。Pipeline 主要对数据进行清洗、校验、转换然后进行摄入。Pipeline 底层是由 Flink 实现的，其内部又按照功能划分成 Preprocessor 和 RawEvent Processor。其中，Preprocessor 进行数据清洗、校验并将结果输出给 RawEvent Processor，然后进行转换。

事件转换规则可以定义为

$$\text{Matcher} \rightarrow \text{Action}$$

即如果事件可以和 Matcher 匹配，那么会对这个事件应用 Action。其中，Matcher 是一个筛选条件，由前面描述的 QL 定义，通过 Matcher 可以筛选出一批事件。

Action 通过 grok 表达式定义，即如果日志内容与某个 grok 表达式匹配，会从中提取字段，并将这些字段赋值给这个事件。Action 的示例如下

%{notspace:domain} %{integer:tax} %{data:type}

系统会维护一个由用户定义的规则列表，每条原始事件会按照预定义的顺序依次应用每一条转换规则，直到第一个匹配的规则。通过这种方式可以从原始事件中，富化出很多的字段，实现事件的动态解析。

按照如上处理后的事件统一称为 well-formed event，会被再次写入 Kafka 集群，等待被下游的 Writer Proxy 消费。

6) Writer Proxy

Writer Proxy 也是基于 flink 实现的，划分成2个 Job：Router Job 和 Write Job。为了提升查询性能以及压缩率，同时实现故障隔离，Router Job 会按照一定规则将 well-formed event 路由到不同的 Writer Job。在底层 Event Storage 是由多个标准集群组成的大集群，标准集群和 Write Job 之间是一对一的映射关系，实现一对一的写入。当某一个标准集群故障，出现时延只会影响部分数据，降低了影响范围。

Writer Job 为了提升吞吐，也是按照 Batch 的方式将一批事件数据进行写入。同时，为了写入链路的健壮性，Writer Job 还实现了类似 TCP 拥塞控制的限流机制以及防止雪崩的熔断机制。

7) Event Storage

Event Storage 用于存储事件数据，提供高吞吐的写入以及低时延查询。如前所述，为了进行故障隔离，Event Storage 会划分为多个标准集群，具体的实现细节见后文。

2.1.2 查询侧

相对于写入侧，查询侧会简单一些，主要由Query Proxy、Storage Meta和应用程序接口（API, application programming interface）构成。

1) API

API层以Restful API的形式提供服务，具有鉴权、数据查询等能力。同时，API侧由于贴近用户，会提供更符合业务需求的查询接口，然后将这些查询请求转换为更抽象、通用的请求结构来查询后端。

2) Storage Meta

元数据服务，用于存储集群以及索引元数据。集群元数据可以进行集群的剪枝，根据路由信息将请求转发到指定的集群，避免全集群的广播，缩短请求耗时。索引元数据包括：时间以及其他路由字段的分布。通过索引元数据对请求需要扫描的索引进一步剪枝，提升查询性能。

3) Query Proxy

Query Proxy用于对查询请求的封装，提供基础的、通用的查询能力。将内部定义的查询QL转换为事件存储引擎的领域查询语言（DSL, domain specific language）。同时会实现如下复杂功能。

① 聚合缓存。由于观测事件符合时序特征，同时数据只会新增，不会变更。因此，在聚合场景下非常适合做缓存，从而缩短请求的响应时间并且降低对底层事件存储引擎的查询压力。

② Search After。事件搜索时支持翻页。

③ 采样。支持搜索、聚合分析时的固定条数采样以及固定比例采样。

④ Union。对2个查询结果取并集。

⑤ 任意字段排序。由于事件符合时序特征，因此事件存储引擎的主要优化是针对时间排序的，其他字段排序性能会急剧下降。通过在Query Proxy实现一定的策略优化，可以将任意字段排序耗时降至合理的范围。

通过Query Proxy将底层实现的复杂性封装，在API和底层事件存储引擎构建一个稳定的接口，可以很好地隔离底层引擎优化演进以及上层业务快速迭代带来的摩擦成本，从而大幅提升研发效率。

2.2 事件存储引擎的实现

如果要针对上述事件模型进行存储实现，在不考虑性能以及扩展性的情况下，可以有多种存储选型，如：关系数据库MySQL、在线分析处理型（OLAP, online analytical processing）计算引擎Doris、Clickhouse以及分布式搜索和分析引擎Elasticsearch。由于MySQL是传统的关系型数据库，对大数据量存储的支持度不够，同时不能很好地支持全文检索，且聚合分析能力也相对较弱；而Doris和Clickhouse属于大数据存储引擎，具备大规模数据的存储和计算能力，所以聚合分析能力很强，但是对全文检索能力的支持相对较弱。最终本文选择了Elasticsearch作为事件存储引擎的底层实现，主要原因如下：首先，Elasticsearch本身就是一个功能强大的分布式搜索引擎，所以它具备强劲的全文检索能力，而这个能力对于事件引擎来说是至关重要的；其次，Elasticsearch支持列式存储格式，所以它能够提供不错的聚合分析能力；再次，Elasticsearch提供灵活的插件能力，使定制开发得以实现；最后，Elasticsearch经过了十几年的发展，相对于其他的大数据存储引擎来说成熟度更高，意味着运维成本比较低。

Elasticsearch自身提供分布式能力，一个集群由Master、Coordinator和Data节点组成。其中，Master管理集群元数据以及节点信息；Coordinator用于对请求进行协调；Data节点用于数据的存储和计算。集群元数据包括节点信息、索引信息以及索引的字段信息，在集群以及数据规模超过一定程度后，元数据的处理会成为瓶颈，导致故障恢复耗时增加。所以，为了应对庞大的数据规模，即存储以及写入数据规模，本文创新性地提出了集群部署单元化的概念。通过精细的性能测试，确定了性能最佳时的集群规模以及集群配置，这样一个集群称为标准集群。然后将大集群按照标准集群为单元进行拆分。集群拆分后，需要有一个组件用于将请求路由以及进行响应结果的合并，这个组件就是跨集群搜索（CCS, cross cluster search）集群。事件存储引擎架构如图2所示。

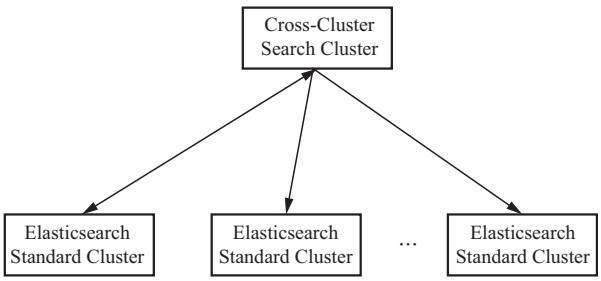


图2 事件存储引擎架构

上述架构会带来以下好处。

① 故障域隔离。当部分集群出现读、写故障时，不会影响其他集群。而之前的广播模式面对这种情形时，会导致整体不可读写，从而将全局性故障降级成局部故障，系统的可用性得到提升。

② 更好的写入能力。因为 Elasticsearch 是广播写，如果集群规模过大，出现慢节点的概率会很高，写入性能很可能受影响。基于标准集群写入，限制了集群规模，降低了写入变慢的概率。

③ 更好的查询性能。可以基于一定策略将数据路由到某个集群，从而缩小查询的节点数，避免了广播查询带来的时延，提升了查询性能。

④ 更低的故障恢复时间。标准集群的元数据更小，在故障恢复时，需要更少的时间恢复元数据，降低了故障恢复的整体时间。

⑤ 更好的可维护性。集群维护重启时，会按照一个标准集群的维度进行，影响面限制在单个集群。而之前大集群时，如果出现问题会导致整体不可用。

按照上述的实现架构，已经能够满足事件存储引擎的大部分需求。此外，本文还做了很多优化工作，以比较完美地解决观测场景下的棘手问题，这些工作会在后续的论文中进行讨论。

2.3　系统实现效果

作为一个可观测系统，需要具备对各种观测数据进行探索、分析的能力，同时可以将不同类型的数据打通，进行关联分析。系统的功能模块首先会按照数据源类型进行划分，在不同数据源之间支持按照关联维度进行跳转，具体模块如下。

1) 日志

日志模块支持对日志搜索、绘图分析（聚合分析）以及请求总览。其中，请求总览可以按照一定的连接字段对关联的日志进行分析，例如，包括相同 trace id 或 order id 的日志。日志模块具体实现如图 3 所示，页面会展示日志的时序分布，同时会按照错误日志进行聚合，便于对错误进行处理。

2) 服务观测

基于 Trace 数据构建了服务观测模块。首先，根据 Trace 数据生成了服务的 RED（request、error 和 duration） 指标，以及用于展示服务调用拓扑图的上下游功能。此外，还包括服务入口分析、基础设施和运行时间指标以及慢 Trace 分析功能。服务观测模块如图 4 所示。

3) 指标模块

指标模块主要用于查询 Metric 数据，会以曲线图形式展示时序的指标数据。后续，系统还会进一步实现指标库，对指标的元数据进行管理，方便用户使用指标。指标模块如图 5 所示。

4) 大盘模块

系统还提供大盘功能，便于用户将常用的指标以及日志配置成大盘，将多个图表展示在一起，使问题排查变得更容易。

3　结束语

本系统部署在中国农业大学数据中心，并已经持续运行，在日常的应用系统运维工作中发挥了巨

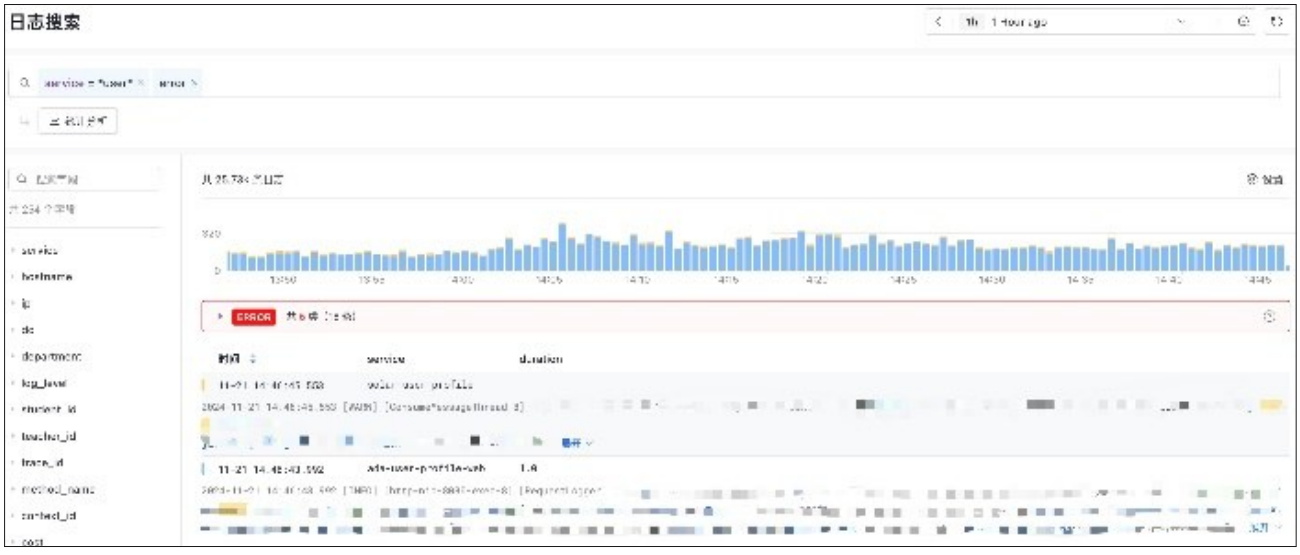


图 3　日志模块具体实现

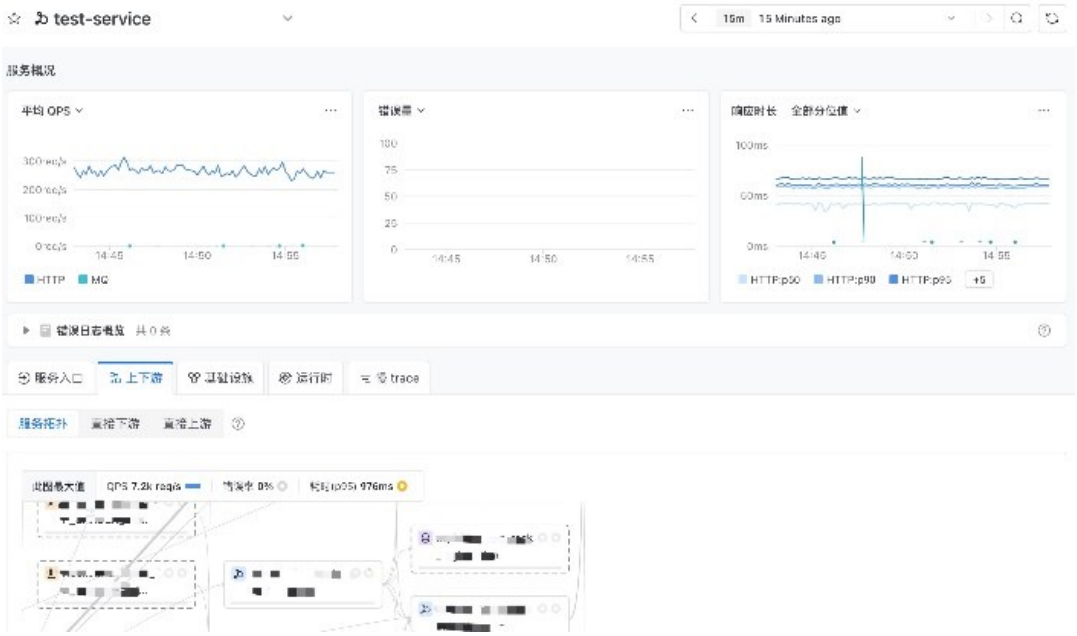


图4　服务观测模块

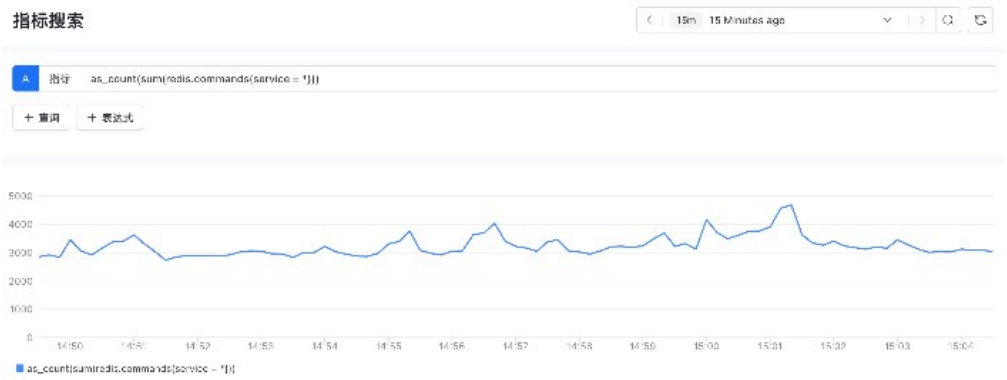


图5　指标模块

大的作用，运维效率得到了大幅的提升。

1) 相对于传统的监控系统（例如 Zabbix[11]、OpenTSDB），本系统除了提供 Metric 数据外，还可以对 Trace 数据和 Log 进行探索和分析。更多的数据源将赋能应用开发人员的排查能力，缩短根因定位时间。

2) 通过实现 Metric 数据、Trace 数据以及 Log 数据的联动以及关联分析，可以大幅提升维护人员的问题排查效率，将一般的、普遍性问题的定位耗时降至 1min 内。

3) 通过提供更多智能的、高级的问题排查手段（例如离群分析、请求总览、突增突降分析等），可以进一步降低问题的定位时间。

4) 通过统一的事件模型存储多种观测数据，改变了每种数据部署一个系统的现状，降低了用户的使用系统的负担，大幅提升了系统的开发、运维效率。

参考文献：

[1] ZHANG Z H, ZHAN J F, LI Y, et al. Precise request tracing and performance debugging for multi-tier services of black boxes[C]//Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems & Networks. Piscataway: IEEE Press, 2009: 337-346.

[2] LAI C A, KIMBALL J, ZHU T, et al. Milliscope: a fine-grained monitoring framework for performance debugging of n-tier web services[C]// Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). Piscataway: IEEE Press, 2017: 92-102.

[3] MI H B, WANG H M, ZHOU Y F, et al. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems[C]//Proceedings of the IEEE Transactions on Parallel and Distributed Systems. Piscataway: IEEE Press, 2013: 1245-1255.

[4] SRIDHARAN C. Distributed systems observability : a guide to building robust systems[M]. Sebastopol: O'Reilly, 2018.

[5] BECKETT D. Combined log system[J]. Computer Networks and ISDN Systems, 1995, 27(6): 1089-1096.

[6] HE P J, ZHU J M, HE S L, et al. Towards automated log parsing for large-scale log data analysis[J]. IEEE Transactions on Dependable and Secure Computing, 2018, 15(6): 931-944.

[7] ZHAO X, RODRIGUES K, LUO Y, et al. Log20: fully automated optimal placement of log printing statements under specified overhead threshold[C]//Proceedings of the 26th Symposium on Operating Systems Principles. New York: ACM Press, 2017: 565-581.

[8] DUMAIS S, JEFFRIES R, RUSSELL D M, et al. Understanding user behavior through log data and analysis[M]//OLSON J S, KELLOGG W A, eds. Ways of Knowing in HCI. New York, NY: Springer New York, 2014: 349-372.

[9] LANDAUER M, SKOPIK F, WURZENBERGER M, et al. System log clustering approaches for cyber security applications: a survey[J]. Computers & Security, 2020, 92: 101739.

[10] SHEKHTMAN L, WAISBARD E. EngraveChain: a blockchain-based tamper-proof distributed log system[J]. Future Internet, 2021, 13 (6): 143.

[11] 廖湘科, 李姗姗, 董威, 等. 大规模软件系统日志研究综述[J]. 软件学报, 2016, 27(8): 1934-1947.
LIAO X K, LI S S, DONG W, et al. Survey on log research of large scale software system[J]. Journal of Software, 2016, 27(8): 1934-1947.

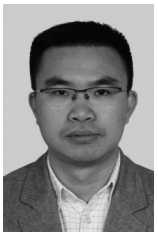
[12] 郑博, 王煜彤, 燕钰, 等. 时间序列数据库管理: 技术、系统与展望[J]. 工业技术创新, 2022, 9(4): 12-21.
ZHENG B, WANG Y T, YAN Y, et al. Management of time series database: technology, system and prospect[J]. Industrial Technology Innovation, 2022, 9(4): 12-21.

[13] YANG Y, WANG L, GU J, et al. Transparently capturing execution path of service/job request processing[M. Cham: Springer International Publishing, 2018: 879-887.

[14] THERESKA E, SALMON B, STRUNK J, et al. Stardust[J]. ACM SIGMETRICS Performance Evaluation Review, 2006, 34(1): 3-14.

[作者简介]

陈红茜（1987– ），女，湖南常德人，博士，中国农业大学高级工程师，主要研究方向为计算机软件系统、数据库技术、校园信息化、农业信息化等。



邱小彬（1977– ），男，福建龙岩人，中国农业大学高级工程师，主要研究方向为计算机技术、校园信息化等。

屈阳（1996– ），女，北京人，中国农业大学助理工程师，主要研究方向为计算机软件系统、计算机网络、校园信息化等。

曹磊（1990– ），女，吉林延边人，中国农业大学中级工程师，主要研究方向为计算机技术、校园信息化等。

王居正（1990– ），男，北京人，中国农业大学中级工程师，主要研究方向为计算机软件系统、数据库技术、校园信息化等。

陈昕（1974– ），男，甘肃白银人，博士，中国农业大学副教授，主要研究方向为农业信息化、教育信息化等。