arXiv:2304.13302v1 [cs.DC] 26 Apr 2023

HiQ——一个声明式、非侵入式、动态且透明的可观测性与优化系统*

专为现代人工智能时代的可观测和高性能软件系统而设计

Fuheng Wu† *Ivan Davchev*‡ *Jun Qian*§

摘要

本文提出了一种名为 HiQ 的非侵入式、声明式、动态且透明的系统，用于跟踪 Python 程序运行时信息，同时不影响运行时系统的性能，也不会丢失可观测性。HiQ 可用于单体系统和分布式系统，以及离线应用和在线应用。HiQ 最初是在优化以 Python 编写的大型深度神经网络（DNN）模型时开发的，但它可以推广到任何 Python 程序或分布式系统，甚至可以支持 Java 等其他语言。我们已实现该系统，并将其应用于深度学习模型生命周期管理系统，在保持生产代码整洁且高性能的同时捕获瓶颈。该实现已开源，地址为：https://github.com/oracle/hiq。

1 引言

识别程序的性能瓶颈，对于单体应用和分布式系统都至关重要。许多此类应用和系统都是用 Python 编写的。Python 是一种流行的编程语言，以语法简洁、库功能强大而闻名。在许多情况下，对于给定的 Python 程序或系统，需要跟踪延迟、内存、输入输出等运行时信息，但由于各种限制，不能修改原始代码。在其他情况下，即使将日志记录和追踪代码插入原始代码中，也不可避免地会导致性能下降。如何衡量这种性能下降是一个问题。此外，代码编写完成并开始运行后，另一个问题是：如何动态调整追踪逻辑，在不丢失系统洞察的前提下获得最佳系统性能。

在本文中，我们提出了一种名为 `HiQ` 的非侵入式、声明式、动态且透明的系统，用于跟踪 Python 程序运行时信息，同时不影响运行时系统的性能，也不会丢失洞察。HiQ 既适用于单体应用，也适用于分布式系统。我们在优化以 Python 编写的大型深度学习模型时发明了该系统，但它可以推广到任何 Python 程序或分布式系统。我们已经实现了该系统，并将其应用于深度学习模型生命周期管理软件 `Gamma`，以发现瓶颈，同时保持生产代码整洁并具备高性能。

2 背景

应用性能监控/管理多年来一直是一个普遍存在的问题，也为许多公司催生了利润丰厚的业务。如今市场上有许多 APM 厂商，例如Oracle APM、Elastic APM、IBM Instana APM、Splunk APM 等。

对于离线性能跟踪，长期以来人们一直使用 Python 内置库 `cProfile`、`profile`、`pstats` 等工具。它们是功能强大的分析工具，但也存在许多缺点，包括计时准确性不足、开销高、无关信息过多、难以定制，并且仅适用于开发目的 [2]。

在在线性能跟踪领域，自 Google 在 2010[4], 发表 Dapper 论文以来，涌现出了许多不同的实现，例如 Zipkin（2012年）、Jaeger（2016年）和 Apache SkyWalking（2018年）。基本上，为了解决在线性能跟踪问题，业界有两种方法。第一种传统方法，也称为 `explicitinstrumentation`，要求开发人员手动将跟踪代码片段插入原始源代码中。第二种方法是使用代理线程对原始代码进行插桩。程序启动时，除了主线程外，还会生成一个代理线程，定期触发线程转储，并将线程快照发送到远程分析服务器。该分

*HiQ 由 OCI 视觉服务团队开发，用于将深度学习模型投入生产。通讯作者：fuheng.wu@oracle.com。邮寄地址：100 Oracle Pkwy, Redwood City, CA 94065, USA。

服务器会存储这些信息、对其进行整合，并重建运行时性能信息，供用户查询[5]。

大多数应用性能管理厂商为 Python 程序提供第一种方法作为解决方案。其中一些厂商（如 Elastic APM）也通过第二种方法提供库。然而，即使采用第二种方法，尽管它不会修改原始代码，但仍需要复杂的配置、至少额外运行一个线程，并且由于客户端与数据收集服务器之间频繁的线程转储和通信，会产生额外的网络 I/O。对于 Python 程序，尤其是像我们案例中的 CPU 密集型应用，由于 Python 语言的设计特点，其开销非常大。此外，从定期生成的线程快照中重构运行时信息，其精确度也不如第一种方法得到的结果。例如，在 Apache SkyWalking 中，持续时间少于 10 毫秒的事件将被系统遗漏（图1） [1]。

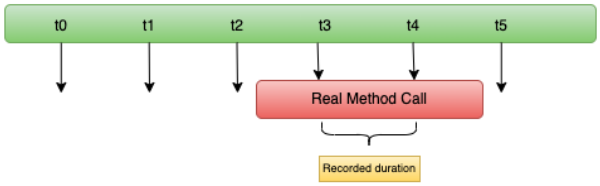

图 1：使用代理时的监控盲区

本发明披露将提出一种新的非侵入式性能跟踪方法，无需额外线程，仅产生极低的网络 I/O 开销，同时还能达到与第一种方法相同的准确度。此外，本发明还提出一种动态控制监控开销的方法，以及一种测量跟踪开销并将其展示给用户的方法。

现有解决方案大多采用上述侵入式 `explicit instrumentation` 方式。现有最先进的解决方案，如 Apache Sky-Walking，采用了基于代理的方法，但存在许多缺陷。

我们提出的方法通过完全不同的方式解决了同一问题。我们首创的解决方案充分利用了 Python 解释器的动态特性。驱动代码首先从外部存储中加载所有相关的变量和函数，这些外部存储可以是配置文件或数据库，并将其载入内存；随后，它再加载目标 Python 代码，可以是脚本或模块。加载目标代码时， `HiQ` 会记录相关变量或函数的原始内存地址。然后，它将动态构造新的函数，以拦截对原始函数或变量的调用。我们提供了多个 Python 装饰器，因此无需额外配置即可轻松追踪延迟、内存和输入输出。我们还提供了接口，供用户定义自己的追踪函数。当这些函数被调用时，内存中会构建调用图，并在图节点中收集和存储数据。另一个控制器进程会设置共享内存中的所有开关，而`HiQ` 驱动代码可以访问该共享内存，因此远程调度器能够控制追踪行为。在分布式环境中，调度器可以针对不同机器将追踪调度到不同级别，例如启用积极追踪、禁用部分或全部追踪，或根据机器负载及其他因素自适应地启用追踪。

我们已将算法实现为一个名为 `HiQ` 的 Python 库。它在不牺牲代码完整性、整洁性和运行时性能的前提下，将应用性能可观测性提升到了新的水平。它使我们的跟踪变得简单、动态、非侵入式、透明且高效。

3 HiQ 方法论

3.1 概述

HiQ 是一套 `declarative、 non-intrusive、 dynamic` 且`transparent`的跟踪系统，适用于**单体**应用和**分布式** 系统。它将运行时信息跟踪提升到了一个你从未想象过的新水平。而且 HiQ 不会在速度和系统性能上妥协，也不会隐藏任何跟踪开销信息。HiQ 适用于输入输出密集型和 CPU 密集型应用。它完全可定制、完全动态且透明。它可以作为开发工具，但设计时也考虑了对生产级应用的支持，因此**你可以在开发和生产环境中使用 HiQ**。此外，它灵活、易用，并且易于与现有开源项目（如 Zipkin、Jaeger、Kafka、服务网格）以及 Oracle APM、Oracle Functions、Oracle T2 等企业级服务集成。

3.2 提出的数据结构和系统架构

3.2.1 驱动代码和目标代码

HiQ 中最基本的概念之一是**驱动代码**和**目标代码**。一个典型的设置可以用图2表示。

目标代码是包含业务逻辑的原始 Python 脚本或模块，可被导入

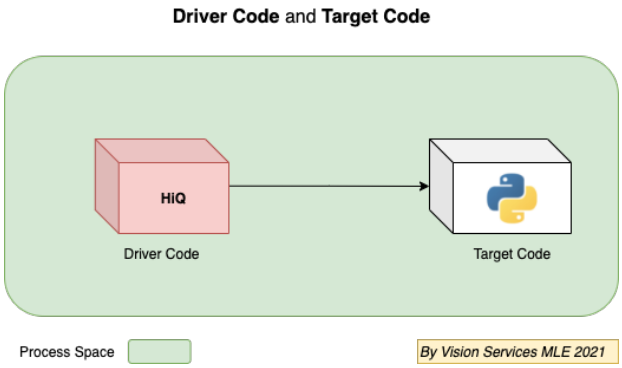

图 2：HiQ 跟踪的基本设置

驱动代码包含跟踪逻辑。它会导入目标代码并调用其中的函数。

3.2.2 使用 MinimumCoding 进行声明

无需深入源代码来插入日志记录和追踪代码，HiQ 使用声明式配置文件来描述您想要追踪的内容。它可以是一个格式如下的 JSON 文件：

```
[{"name": "f1",
"module": "my_model_1",
"function": "func1",
"class": ""}, {"name": "f2",
"module": "my_model_2",
"function": "func2",
"class": ""}, ]
```

JSON 文件包含一个字典列表，用于指定所关注的模块、类和函数，其中name 是 HiQ 树节点的名称。上述 JSON 可以转换为如下指令：追踪模块 `my_model_1` 中名为 `f1` 的函数 `func1`，以及追踪模块 `my_model_2` 中名为 `f2` 的函数 `func2`。

用户只需声明跟踪目标并选择一个 HiQ 类，例如，`HiQLatency` 用于延迟跟踪，`HiQMemory` 用于内存跟踪，随后 HiQ 系统便会自动进行跟踪。

3.2.3 HiQ 树

HiQ 树是一种专门的区间树数据结构，旨在用于性能跟踪和根因查找。对于单体应用，它类似于所谓的 `call graph`[3]。对于分布式系统，它类似于 `trace tree` Google 的 Dapper 论文中所描述的[4]。HiQ 树既适用于单体应用，也适用于分布式跟踪，因此结合了二者的特点`call graph` 和 `trace tree`。图 3展示了 HiQ 的样子。

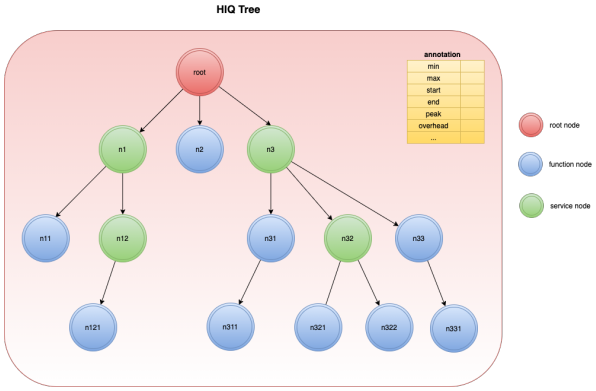

图 3：HiQ 树

HiQ 树的特性包括：

- HiQ 树由两类子树组成：服务子树和函数子树。服务子树由服务节点组成，函数子树由函数节点组成。

- HiQ 树可配置、动态，并可根据用户需求（例如系统性能服务等级协议）进行自适应调优。– 加载到共享内存中的配置文件将控制应追踪或忽略哪些进程和节点，以及在何种条件下执行这些操作。– 树结构可以根据运行时性能信息和配置动态变化。例如，可以将 HiQ 树设置为忽略所有跨度小于 100 毫秒的函数节点。实际上，当函数树设置为 `concise` 模式时，它会忽略所有跨度为零的节点。这对于内存追踪非常有用，因为由于操作系统的内存管理操作或程序的垃圾回收，内存开销并非单调变化。– 树结构可以自适应地变化，以满足系统性能服务等级协议。

- HiQ 树的子树会根据不同的子树类型，以不同方式构建

。HiQ 服务子树会被发送到 HiQ 服务器，之后重新构建该树。相关函数调用完成后，HiQ 函数子树会在运行时于内存中构建。多进程设计降低了开销，并且对用户透明。此设计还通过免除复杂重构算法的需求，简化了树的重构[6]。

•HiQ 树以非侵入式的方式构建或重建，从而将业务逻辑与追踪逻辑分离。这带来了诸多好处和灵活性。例如，我们可以在运行时选择使用不同的日志库，而无需改动目标代码。

3.2.4 以透明开销进行追踪

任何追踪或日志记录都会产生开销。有时开销可能很大，以 Python 的 `profile` 模块为例。开销也取决于代码的编写方式。由于我们讨论的是 Python，因此只关注延迟开销。CPU 和内存开销要么可以忽略不计，要么可以用延迟来表示。许多软件都声称开销极小，却没有说明究竟有多小。你很可能写出了开销巨大的低性能代码，却没有意识到这一点。 `HiQ` **是唯一一个能让用户完全看清这些开销的系统**。HiQ 会分别追踪原始函数和追踪代码的延迟。追踪代码会在原始函数调用前后运行，因此 HiQ 能够轻松计算日志代码的开销。它会将延迟开销附加到每棵 HiQ 树上，让你了解追踪系统本身的影响究竟有多大。延迟开销可以用绝对格式打印，单位为微秒；也可以用百分比格式打印。

```
$ python main_driver.py
func1
func2
[2021-11-01 21:54:18.222424 - 21:54:22.226879]  [100.00%]  _root_time(4.0045)
                                                            [OH:163us]
[2021-11-01 21:54:18.222424 - 21:54:22.226879]  [100.00%]    l___main(4.0045)
[2021-11-01 21:54:18.222472 - 21:54:22.226868]  [100.00%]      l___func1(4.0044)
[2021-11-01 21:54:19.724213 - 21:54:22.226818]  [ 62.50%]        l___func2(2.5026)
```

图 4：HiQ 树中的延迟开销

图 4 展示了开销如何附加在 HiQ 树上的示例。标签 `OH`：根节点下方显示了整棵树的延迟开销。我们可以看到， `main` 函数耗时 4.0045 秒完成，而跟踪开销为 163 微秒，即 0.004%。

3.2.5 HiQ 单体应用架构

对于单体应用，HiQ 驱动代码会在主进程的主线程中启动目标代码。同时，根据您的配置，它还会生成多个进程，用于在本地保存日志，或将指标发送到 HiQ 服务器或 Oracle APM、Kafka 等云服务。系统架构如图 5所示。

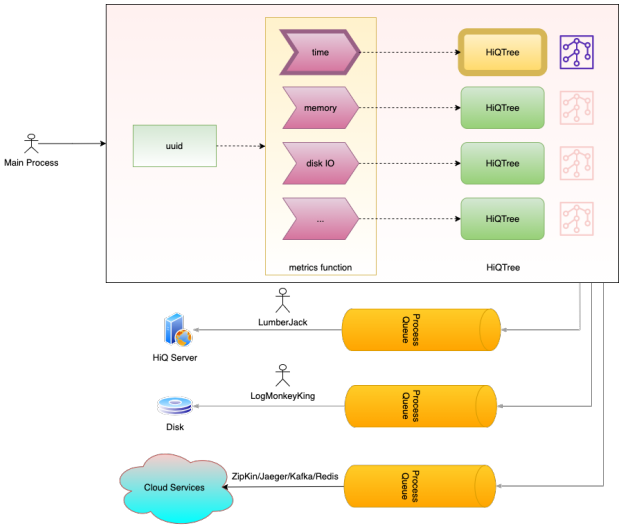

图 5：HiQ 在单体应用中的应用

在主进程中，使用映射数据结构来存放不同指标（包括延迟、内存和磁盘 I/O）对应的不同 HiQ 树。该映射有最大容量限制。当映射大小达到最大值时，HiQ 树会通过 Python 多进程队列发送到其他进程进行进一步处理，并将映射重置为空。

3.2.6 HiQ 分布式系统架构

对于分布式系统，在应用层，HiQ 系统与上文所述相同。但从更高层次的分布式系统角度来看，HiQ 架构如图所示6。

这里假设每个应用都运行在托管于虚拟机（VM）中的 Docker 实例内。这种配置在由 Kubernetes 或 Mesos 管理的分布式系统中非常常见。例如，在 Kubernetes 中，布局完全相同，只是名称不同：虚拟机称为 `node`，Docker 实例称为 `pod` 或 `container`。

首先，用户在网页浏览器中为 HiQ 跟踪设置配置。数据会写入一个 c

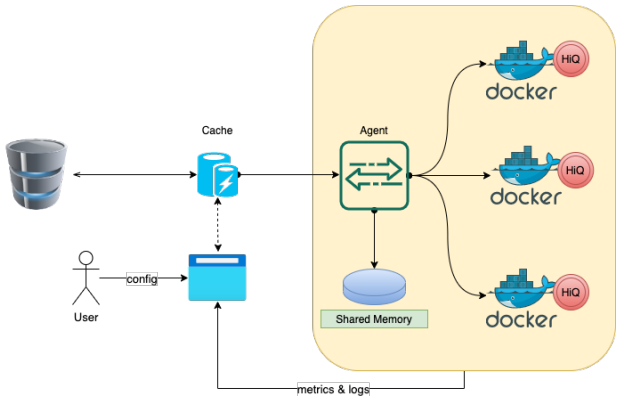

图 6：分布式系统中的 HiQ

系统，然后持久化保存到数据库中。在虚拟机中运行的代理会在配置变更与该虚拟机相关时收到缓存服务器的通知。随后，代理会拉取变更并更新共享内存。该虚拟机中运行的所有 Docker 实例都会使用这块共享内存。配置变更后，HiQ 系统会相应地应用变更。通过这种方式，用户可以在运行时按各个层级控制 HiQ 跟踪：虚拟机、实例、服务和函数。

3.3 用例示例

3.3.1 使用 Oracle APM 进行非侵入式监控的 HiQ

OCI 应用性能监控 (APM)是一项服务，可深入了解应用程序的性能，使 DevOps 专业人员能够快速诊断问题，从而提供一致的服务水平。**HiQ 开箱即支持 OCI APM**。

在 HiQ 中使用 OCI APM 有两种方式。传统方式是使用 `HiQOciApmContext`，其底层使用 `py_zipkin`。这与 Oracle 官方文档中所述的方式相同。现代方式是使用 `HiQOpenTelemetryContext`，它使用新的 `OpenTelemetry` SDK API。以下是使用 HiQ 向 OCI APM 发送指标的示例代码；你甚至不需要显式编写任何发送指标的代码，因为 HiQ 已自动为你完成了这项工作。你需要做的只是声明感兴趣的指标、模块、类和函数，剩下的繁重工作都由 HiQ 完成。

运行此图 7 中的代码将指标发送至 OCI APM，并显示调用图，如图 8所示。

HiQ 简化了向 OCI 发送指标的流程

```python
import hiq
import os

from hiq.vendor_oci_apm import HiQOciApmContext

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with HiQOciApmContext(
        service_name="hiq_doc", span_name="main_driver",
    ):
        _ = hiq.HiQLatency(f"{here}/hiq.conf")
        hiq.mod("main").main()


if __name__ == "__main__":
    os.environ["TRACE_TYPE"] = "oci-apm"
    run_main()
```

图 7：HiQ 与 Oracle APM 集成——代码示例

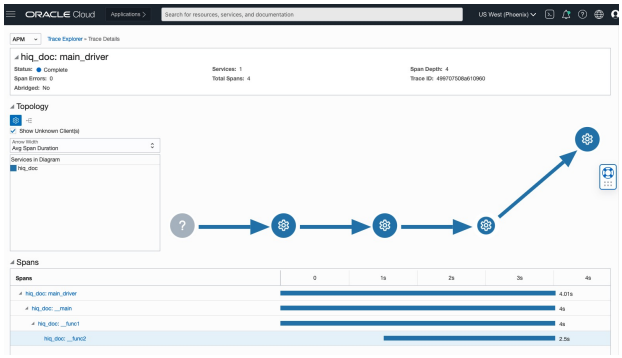

图 8：HiQ 与 Oracle APM 集成——跟踪结果

应用性能管理，而且它比 Oracle 官方应用性能管理文档提供的当前方法更有优势。

3.3.2 HiQ 在深度学习模型生命周期管理系统中的应用

HiQ 已广泛应用于深度学习模型生命周期管理系统，用于分析我们的深度学习模型性能。在分析网页门户中，我们使用 HiQ 进行模型延迟分析，这是 HiQ 在单体应用中的一种实现方式，其中我们将深度学习模型视为单体软件。图 9 展示了一个示例深度神经网络模型的 HiQ 延迟树。我们可以看到每个子模型的时间开销绝对值和百分比。我们可以清楚地看到，文本检测模型占总时间的 24%，是瓶颈。

在分析平台中，我们以分布式方式使用 HiQ，通过这种方式，可观测性中心能够动态启用和禁用不同指标、模块和函数的日志记录，如图 10所示。 `LogMonkey` 控制台会相应地更改运行时日志输出。与市场上的所有其他产品不同，可观测性中心会显示用户选择日志指标所产生的性能开销，使用户能够在性能开销与运行时信息之间找到最佳平衡。

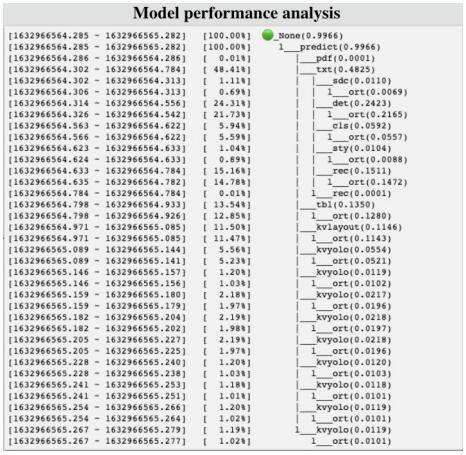

图 9：HiQ延迟树

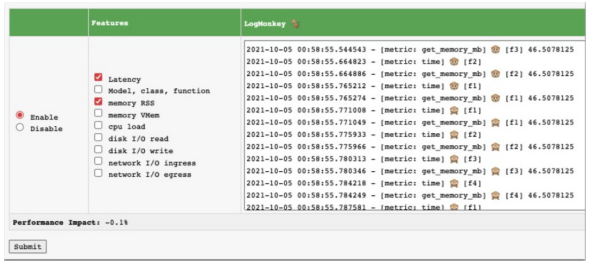

Fig. 10：分布式系统中的动态设置 tem

3.3.3 HiQ与Jaeger集成进行分布式跟踪

Jaeger 受 Dapper 和 OpenZipkin 启发，是由 Uber Technologies 创建并捐赠给云原生计算基金会的分布式追踪平台。它可用于监控基于微服务的分布式系统。截至本文撰写时，Jaeger 支持两种序列化协议：Thrift 和 Protobuf。HiQ 可与 Jaeger 及这两种协议无缝集成。

图 11是通过 Thrift 和 HTTP 向 Jaeger 发送指标数据的驱动代码。与图 7相比，您可以看到唯一的变化在第 4 行和第 10 行。您只需添加一个上下文管理器 `hiq.distributed.HiQOpenTelemetryContext` 即可使 Jaeger 追踪正常工作。

运行驱动代码并检查 Jaeger用户界面，我们可以看到图 12 中的追踪记录已被记录。

除上述使用场景外，HiQ 还开箱即用地支持许多框架，例如 Zipkin、Flask、FastAPI 等。

```
1  import os
2
3  import hiq
4  from hiq.distributed import HiQOpenTelemetryContext, OtmExporterType
5
6  here = os.path.dirname(os.path.realpath(__file__))
7
8
9  def run_main():
10     with HiQOpenTelemetryContext(exporter_type=OtmExporterType.JAEGER_THRIFT):
11         driver = hiq.HiQLatency(f"{here}/hiq.conf")
12         hiq.mod("main").main()
13         driver.show()
14
15
16 if __name__ == "__main__":
17     run_main()
```

图11：HiQ与Jaeger集成 - 代码示例

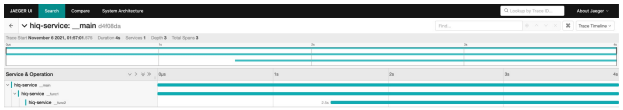

图12：HiQ与Jaeger集成 - 跟踪结果

4 结论

在本文中，我们提出了 HiQ，用于提供一种声明式、非侵入式、动态且透明的方法，以追踪 Python 程序的性能。它既适用于单体应用，也适用于分布式系统追踪，并且可以轻松集成到现有软件或服务中。HiQ 可以作为现有应用性能管理产品的一项新功能嵌入其中，也可以作为一种全新的性能监控、根因分析和系统日志记录方式。我们已在开发、预生产和生产环境中使用 HiQ，它帮助我们发现了错误并优化了服务。

参考文献

[1] “Apache SkyWalking：使用性能分析修复分布式跟踪的盲点”。载于：skywalking.apache.org（2021）。网址：
h
t
t
p
s://
s
ky
walking.apache.org/blog/2020-04-13-apache-skywalking-profiling/#the-limitations-of-the-profile-method。[2] Emery D. Berger。“Scalene：面向 Python 的脚本语言感知型性能分析工具”。载于：CoRR abs/2006.03879（2020）。arXiv：2006.03879。网址：https://arxiv.org/abs/2006.03879。[3] David Grove 等。“面向对象语言中的调用图构建”。载于：第 32 卷。1997 年 10 月，第 108–124 页。doi：10.1145/263700.264352。[4] Benjamin H. Sigelman 等。Dapper：大型分布式系统跟踪基础设施。技术报告。Google 公司，2010 年。网址：https://research.google.com/archive/papers/dapper-2010-1.pdf。[5] “使用 Oracle 应用性能监控”。载于：Oracle® Cloud（2021 年 2 月），第 92 页。网址：https://docs.oracle.com/en/cloud/paas/manag

ement-cloud/apmcs/using-oracle-application-performance-monitoring.pdf。
[6] Sheng Wu。 *STAM*：增强高度分布式和大规模应用系统的拓扑自动检测。技术报告。Apache SkyWalking，2020 年。 网址： https://wu-sheng.github.io/STAM/。