Cast：面向生产云服务系统的自动化韧性测试

Zhuangbin Chen 中山大学软件工程学院 中国广东珠海
chenzhb36@mail.sysu.edu.cn

Zhiling Deng 中山大学软件工程学院 中国广东珠海
dengzhling3@mail2.sysu.edu.cn

Kaiming Zhang 中山大学软件工程学院 中国广东珠海
zhangkm7@mail2.sysu.edu.cn

Yang Liu 中山大学软件工程学院 中国广东珠海
liuy2355@mail2.sysu.edu.cn

Cheng Cui 华为云 中国广东深圳
cuicheng2@huawei.com

Jinfeng Zhong 华为云 中国广东省深圳市
hongjinfeng@huawei.com

Zibin Zheng*中山大学软件工程学院 中国广东省珠海市
zhzibin@mail.sysu.edu.cn

arXiv:2602.00972v1 [cs.SE] 1 Feb 2026

摘要

微服务架构的分布式特性带来了显著的韧性挑战。传统测试方法受限于大量人工工作和过度简化的测试环境，无法捕捉生产系统的复杂性。为解决这些局限，我们提出 Cast，这是一个用于生产环境微服务韧性测试的自动化端到端框架。它通过将生产流量重放至涵盖全面应用级故障的故障库中，触发内部错误处理逻辑，从而实现高保真度测试。为管理组合测试空间，Cast 采用复杂度驱动策略，系统地精简冗余测试，并优先执行针对最关键服务执行路径的高价值测试。Cast 通过三阶段流水线（即启动、故障注入和恢复）自动化测试生命周期，并使用多维预言机，根据细致的标准自动验证系统韧性。Cast 已在华为云部署八个多月，许多服务团队已采用它来主动应对韧性漏洞。我们对四个拥有数百万条跟踪记录的大规模应用程序进行分析，发现了 137 个潜在漏洞，其中 89 个得到开发人员确认。为进一步量化其性能，我们使用包含 48 个复现缺陷的基准测试集对 Cast 进行评估，其覆盖率高达 90%。结果表明，Cast 是一种实用且有效的解决方案，可系统地提升工业微服务系统的可靠性。

CCS 概念

• **软件及其工程** → **软件测试与调试**；• **计算机系统组织** → **可靠性**。

关键词

微服务、软件可靠性、韧性测试

ACM 参考格式：

Zhuangbin Chen、Zhiling Deng、Kaiming Zhang、Yang Liu、Cheng Cui、Jinfeng Zhong 和 Zibin Zheng。2026 年。Cast：面向生产云服务系统的自动化韧性测试。载于*2026 IEEE/ACM 第 48 届国际软件工程会议（ICSE-SEIP '26），2026年4月12日至18日，巴西里约热内卢。* ACM，纽约州纽约市，美国，12 页。https://doi.org/10.1145/ 3786583.3786867

1 引言

微服务架构的广泛应用，从根本上改变了现代云应用的设计和部署方式。这种架构范式通过将单体应用分解为松散耦合、可独立部署的服务，在敏捷性、可扩展性和可维护性方面带来了显著优势。然而，微服务系统高度分布式的特性也带来了系统韧性方面的重大挑战系统韧性，尤其是在云环境中面对复杂且往往具有非确定性的故障时 [20, 51]。

韧性测试对于确保微服务应用的可靠性至关重要 [25, 38, 55]。不同于侧重于功能正确性的传统测试方法 [7, 11, 30]，韧性测试评估软件系统在底层故障条件消除后，容忍、适应并从故障中平稳恢复的能力。这在生产环境中尤为重要，因为许多故障都是暂时性的 [18, 28, 29]。诸如临时网络拥塞或资源竞争等问题，可能导致服务减速或间歇性错误，而这些问题可能会自行恢复。

*Zibin Zheng 是通讯作者。

本作品采用知识共享署名-非商业性使用-禁止演绎 4.0 国际许可协议授权。
ICSE-SEIP '26，巴西里约热内卢 ©2026 版权所有者/作者所有。ACM ISBN 979-8-4007-2426-8/2026/04 https://doi.org/10.1145/3786583.3786867

一个具有韧性的系统必须能够承受这种暂时的性能下降，而不会造成全面中断。此外，微服务系统相互关联的特性意味着，单个组件发生故障可能会沿着依赖链传播，进而影响多个服务。在这种情况下，故障组件修复后，受影响的服务必须能够自主恢复到健康运行状态。

尽管微服务韧性测试至关重要，但要有效开展这类测试仍面临重大挑战。传统方法在两个关键方面存在局限。首先，这些方法通常需要投入大量人工工作 [27, 49, 54]，，因此不适用于大规模系统。这包括手动设计测试用例、配置复杂的故障注入场景，以及定义超越简单状态检查的适当断言。这样的人工流程速度缓慢、容易出错，也无法适应现代服务的复杂性及其快速演进。其次，由于故障模型和验证机制的表达能力不足，这些方法在工业系统中的适用性有限。许多研究 [16, 23, 52] 依赖规模较小的基准（例如 TrainTicket [22, 58]，和 SockShop [50]），无法体现生产环境的规模和复杂依赖关系。注入的故障通常较为通用（例如基本的 HTTP错误），并通过简单检查（例如状态码 200）进行验证，这使测试人员难以创建自定义检查。这些方法无法涵盖现实世界中丰富多样的故障，例如特定的中间件异常或复杂的异步问题 [15, 37, 48]。

为解决这些局限性，我们提出 Cast，这是一个端到端框架，可在工业环境中开展可扩展且有效的微服务韧性测试。我们的方法有三个关键设计目标。首先，为实现**生产级保真度**，Cast 将在线流量录制与回放和模拟技术相结合，以进行细粒度的应用级故障注入。为确保录制的流量能够成功重放，[35]，Cast 会动态识别并更新状态依赖变量，以反映当前的执行上下文。此外，Cast 还构建了一个包含真实故障模式（包括软件级异常和通信错误）的故障库，以严格检验服务的内部错误处理逻辑。其次，为实现**可扩展性**，Cast 采用复杂度驱动策略，应对测试空间的组合爆炸。它会智能地选择一组数量精简但效果显著的测试用例，针对最复杂且易发生故障的执行路径。这为主动发现关键韧性漏洞提供了系统化解决方案。最后，为实现**自动化**，Cast 通过三阶段流水线（即启动、故障注入和恢复）统筹整个测试执行生命周期，同时自动依据多方面标准验证系统韧性。

我们已在华为云部署 Cast 超过八个月，为大量微服务应用提供自动化韧性测试服务。这些应用涵盖存储、网络、数字平台和物联网等多个业务领域。通过长期部署，我们深入了解了大规模韧性测试在实践中面临的挑战与权衡。本文报告了四个大型且具有代表性的应用的测试结果，这些应用由数百个微服务组成，具有流量大、架构复杂、服务依赖关系错综复杂以及技术多样等特点。总体而言，Cast 发现了 137个潜在漏洞，其中 89 个在撰写本文时已由开发人员通过人工调查确认。此外，在一项针对包含 48 个复现的缺陷的基准的受控实验中，Cast 的检测覆盖率达到 90%。这些结果表明，我们的框架能够有效提升工业环境中微服务系统的可靠性。总之，我们的主要贡献如下：

- 我们提出 Cast，一个用于微服务韧性测试的自动化端到端框架，它将基于重放的故障注入与复杂度驱动策略相结合，以优先处理易发生故障的执行路径并剪除冗余测试。Cast 编排整个测试生命周期，并采用多维预言机，自动根据最终结果和内部健康状态评估弹性。
- 我们通过 Cast 在华为云为期八个月的大规模部署展示了其实用效果，许多开发团队已在使用它。我们报告了其长期有效性，以及在复现漏洞上的实验结果，从而验证了其现实影响。
- 我们分享了从工业部署中获得的关键经验，涉及超越API边界的深度验证预言机的重要性、完美测试优先级排序与自动化效率之间的权衡，以及集成到现有开发工作流中的实际考量。

2 背景

2.1 微服务系统

微服务架构风格已成为构建大规模云应用的事实标准 [33]。与单体应用程序不同，微服务系统由一组小型自治服务组成，每个服务负责特定的业务能力。这些服务可独立开发、扩展和维护，为组织带来更高的敏捷性与灵活性。它们通常通过网络使用 HTTP/REST 或 gRPC 等轻量级协议相互通信。系统的整体行为源于这些服务之间错综复杂的交互，因而难以推理和预测。单个服务中的故障可能引发级联故障并在整个系统中传播，从而导致大范围服务中断。

微服务并非孤立运行。它们高度依赖由平台组件构成的共享骨干，以管理状态、通信和数据持久化。这包括用于持久化存储的数据库 （例如 MySQL、MongoDB）、用于异步通信和事件驱动工作流的消息代理 （例如 Kafka、Pulsar），以及用于提升性能、降低延迟的内存缓存 （例如 Redis、Memcached）。尽管这些组件在设计上具备高可用性，但它们与微服务之间复杂的交互会带来显著的错误传播风险。应用程序处理组件不可用、连接错误或数据序列化问题的能力，对其整体弹性至关重要。因此，现代微服务系统构成了一个由业务服务和基础设施组成的相互依赖网络，其弹性取决于每个组件及每次交互的稳健性。

2.2 微服务系统的韧性测试

弹性是指系统承受故障并维持可接受服务水平的能力。对于微服务系统，这

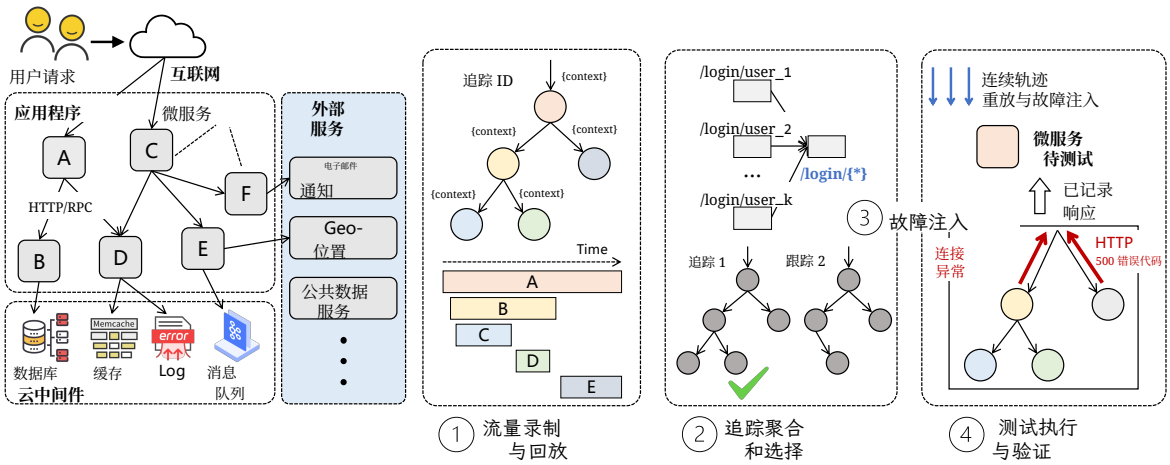


图1：总体框架　　Cast 用于自动化微服务韧性测试的框架　　韧性测试

意味着要妥善处理服务、网络和平台组件不可避免的故障。韧性测试（在混沌工程这一术语的推广下广为人知）旨在通过注入故障，主动发现漏洞，从而增强人们对系统在多变生产环境中存续能力的信心。根据故障注入的粒度，现有方法大致可分为两类。基础设施级方法 [52] 侧重于操控环境，例如终止虚拟机、引入延迟或丢弃数据包。尽管这些故障能有效检验针对连接问题的重试机制和熔断器等基本弹性策略，但它们的粒度通常过粗，无法触发与内部服务状态相关的应用特定错误处理逻辑。相比之下，应用级方法 以业务逻辑为目标，用于验证服务降级机制或恢复流程。例如，模拟特定的业务级异常（如部分 API 故障或平台错误）就需要更精细的粒度。

为实现高保真的应用程序级测试，研究人员和从业者通常会采用两种常见且往往互补的技术，即流量录制与回放和模拟。录制回放通过捕获并重放生产流量来生成逼真的工作负载，确保测试场景能够反映真实的用户交互。模拟则通过将下游依赖替换为可返回预设故障的模拟版本，提供精细控制。然而，[9, 10, 36, 45, 46] 利用这些技术的方法通常面临以下关键挑战：

- **状态依赖**：生产流量具有状态性且对时间敏感。直接重放录制的请求可能会因数据过期而失败，例如时间戳过期、一次性幂等键或无效令牌。有效的重放系统必须智能地识别并刷新这些动态变量，以反映当前测试环境，确保请求有效。

- **可扩展性**：生产流量的规模极其庞大，无法重放每个请求。此外，潜在故障注入点的数量（例如，每次服务间调用、每次数据库查询）会导致可能的测试用例数量组合爆炸。简单或随机地选择测试既低效，也可能遗漏关键但不常见的执行路径。

- **自动化**：要实现测试生命周期的全面自动化，在运维方面是一项重大挑战。这需要复杂地协调测试环境的搭建、工作负载的启动以及故障注入的同步。另一个核心难题是测试预言机问题，即如何自动、可靠地判断测试是通过还是失败。

3 方法论

3.1 概述

为应对这些挑战，我们提出了 Cast，这是一个面向基于 Java 的应用程序、用于有效开展微服务韧性测试的自动化框架。Cast 的总体框架如图 1 所示，包含四个阶段，即 流量录制与回放、轨迹聚合与选择、故障注入和测试执行与验证。在第一阶段，Cast 捕获生产环境流量，并采用轻量级启发式方法识别状态依赖变量并为其设定参数，从而将原始跟踪转换为可重放的测试模板。为应对海量流量，第二阶段采用复杂度驱动策略，对相似的用户交互进行聚合，并选择一小组测试用例，以覆盖最易发生故障的执行路径。第三阶段包括设计一个全面且逼真的应用级故障库，以及一种有原则的剪枝机制，用于识别影响最大的注入目标。最后一个阶段通过三阶段流水线和多方面的验证预言机，自动执行整个测试流程，以评估系统韧性。

3.2 流量录制与回放

Cast 首先记录生产流量，并将其重建为由具有因果关系的跨度组成的分布式跟踪。虽然 OpenTelemetry [43] 等行业标准提供了分布式追踪能力，但它们主要是为被动监控而设计的。Cast 不仅需要执行录制，还需要主动进行运行时干预，以支持流量回放和故障注入（例如，拦截数据库调用并返回模拟响应或抛出异常）。因此，我们基于动态面向切面编程框架实现了一种非侵入式插桩技术，

使用 Java 代理 [2, 34, 35]。这样一来，我们便能在运行时拦截方法调用，而无需修改源代码，并捕获全面的操作，包括服务间通信（HTTP/RPC）以及与平台组件的交互，例如数据库（通过 JDBC）、消息代理（例如 Kafka、Pulsar）和内存缓存（例如 Redis）。这些跟踪是后续分析、高保真重放和韧性测试的基础产物。

然而，正如第 2.2 节所述，直接重放跟踪往往会因时间敏感变量和状态依赖变量而失败。例如，包含过期时间戳的请求可能会被拒绝。此外，许多 API调用都包含幂等键，以防止重复处理。重放带有过期密钥的请求会被忽略。Liu 等人 [35] 通过将这些变量归类为系统状态、内部状态和外部状态来研究这一问题，并提出了一种基于静态污点分析识别这些变量的方法。然而，这种方法无法扩展到大型服务代码库，并且与所采用的特定框架紧密耦合 [35], ，即 SOFA/SOFABoot。为解决这一问题，我们引入了一种轻量级动态变量识别技术，可直接处理记录的请求和响应负载。我们方法的核心是一种两阶段启发式方法，旨在无需分析源代码即可区分动态变量与静态数据：

- **跨度内相关性**：对于每个记录的跨度，我们首先分析其请求和响应负载，以找出在两者中原样出现的令牌。其基本思路是，许多依赖状态的变量（例如会话 ID 或事务令牌）通常会在请求中收到，并直接传递到响应中以保持上下文。
- **跨度间变化性**：对于通过跨度内相关性识别出的候选令牌，我们随后会比较同一操作的多个跨度中的候选令牌。只有当某个令牌的值在这些独立实例之间有所不同时，才会确认它是动态变量。

图2通过同一逻辑操作记录的两个不同跨度，直观展示了整个过程。在第一阶段，跨度内关联，我们会识别出候选令牌，例如 session_id，它们同时出现在单个跨度的请求和响应中。在第二阶段，

跨度间变化性，我们会在相互独立的跨度之间比较这些候选项。我们观察到 session_id 会发生变化（例如，“f7k9q2”与“r4m8p1”），这证实了它是动态的。相反，domain_id 和 status 等字段在两个跨度中保持不变，因此被视为静态数据。满足这两个条件的令牌会被抽象为参数化占位符（例如 ${*}），从而将录制流量转换为可复用的流量模板。在重放期间，系统会根据测试的执行上下文自动实例化这些占位符：session id 等唯一标识符会获得新生成的值，而时间戳则会替换为当前系统时间。这种以负载为中心的方法将服务视为黑盒，因此与静态污点分析相比，它具有更强的可扩展性，且与框架无关。

3.3 轨迹聚合与选择

建立可重放的流量后，下一步是为韧性测试选取一组具有代表性的跟踪，以应对两个可扩展性方面的挑战。首先，生产环境中的跟踪数量极其庞大

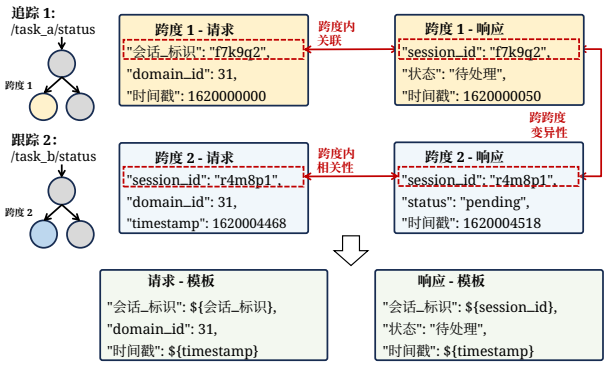


图 2：流量回放中的跟踪模板化

例如，生产级应用可能产生数百万个请求，使得穷举测试不切实际。这些海量跟踪主要源于冗余：对同一接口的重复调用仅在参数值上有所不同。例如，不同用户登录时可能生成数千条不同的跟踪，但它们都调用同一个 POST /api/login/{username} API。因此，为避免冗余测试，我们进行流量聚合，根据入口接口将整个跟踪语料库划分为不同类别。具体而言，我们采用 Drain [26], ——一种常用于日志解析的算法——动态学习根跨度请求行中的静态部分和可变部分，其中通常包括 HTTP 方法和 URI 路径。此步骤能够有效地将海量原始调用提炼为一组不同的服务接口。

即使进行了流量聚合，我们仍面临另一个可扩展性问题：生产环境中的应用程序通常包含数十甚至数百个微服务，每个微服务都提供多个接口。要穷举重放每条唯一轨迹，以测试所有接口，仍然不可行。我们不采用随机抽样，而是设计了一种基于启发式的评分机制，用于选择更有可能暴露弹性问题的部分轨迹。虽然数据血缘驱动的故障注入（LDFI）[5]等复杂算法确实存在，但它们需要详尽的系统模型和静态分析；在我们这种拥有数百个快速演进服务、由多个团队共同维护的多语言环境中，这些要求并不切实际。我们的方法首先通过三个因素的加权组合来量化轨迹复杂度：以跨度数量衡量的轨迹长度、组件的多样性（即涉及的唯一服务和基础设施组件），以及轨迹执行的端到端持续时间。权重可根据具体测试优先级进行配置。涉及组件种类更多、执行路径更长的轨迹会获得更高的复杂度分数，因为它们会触发更多交互，且本质上更容易发生故障。单条轨迹复杂度分数是我们下一层选择的基础，用于确定微服务接口的优先级。接口的复杂度可能因输入参数和业务逻辑而有很大差异，从而产生不同的执行路径。为了全面衡量接口的典型复杂度，我们通过对其所有关联轨迹的复杂度分数取平均值，计算出该接口的聚合分数。

为每个接口分配复杂度分数后，我们采用两级选择策略来确定最终的测试用例集合。首先，根据聚合分数对所有接口进行排序，并选出排名前 K 的候选接口。

其次，对于每个候选接口，我们选取一个最具代表性的测试用例，即个体复杂度分数最高的跟踪。该策略将测试工作集中在系统中最复杂的接口及其最具挑战性的执行路径上，从而为后续故障注入阶段生成一组规模小但效力强的跟踪。

3.4 端点覆盖率引导的故障注入

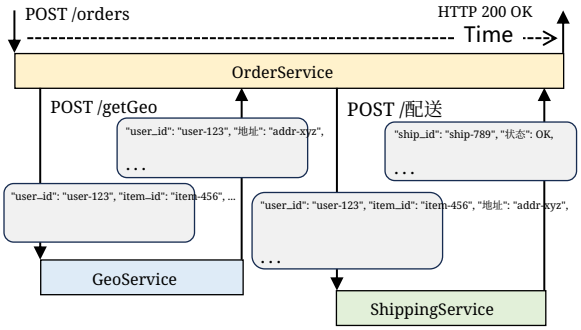
3.4.1 故障库构建。我们的故障注入以一个全面且可扩展的故障库为基础，该库收录了真实的故障模式。具体而言，我们专注于软件级故障，以测试应用程序内部的错误处理逻辑，而非粗粒度的基础设施故障。通过动态插桩，我们创建可注入的故障模块，用于模拟以下两大类故障：

- 平台组件异常：此类别用于模拟不同平台组件的客户端库常见抛出的异常。为构建全面的列表，我们进行了系统性分析，综合采用以下方法：1）全面审查官方 API 文档，以识别其中记载的故障模式；2）分析历史事件，优先关注影响较大的错误；3）采访资深开发人员和 SRE工程师，以获取领域专家知识。由此，我们构建了丰富的故障库，涵盖多种平台组件类别，例如数据库（如 SQLTimeoutException）、消息代理（如 SerializationException）和缓存（如 RedisConnectionException）。该故障库并非一成不变，而是持续更新，以反映技术栈的变化并纳入新发现的故障模式。

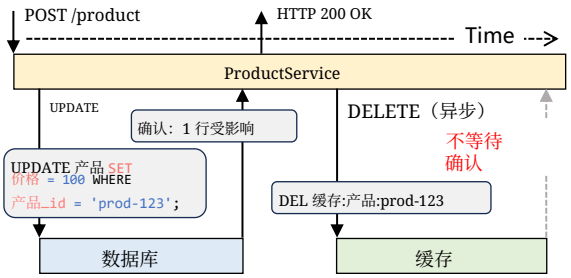
- 服务间通信错误：此类别通过拦截传出的请求来模拟同步通信（例如 HTTP/RPC）故障，以模拟各种中断情况。其中包括引入网络延迟（延迟），以测试缓慢的依赖关系；或抛出 java.net.SocketTimeoutException 等协议级异常，以模拟连接完全失败。此外，还可以不进行实际的网络调用，而是返回经过全面操控的故障响应。我们的库为此提供了广泛的覆盖率，

  允许修改整个响应，包括标准 HTTP 状态码，例如客户端错误（如 401 未授权）和服务器错误（如 500 内部服务器错误、504 网关超时），以及响应正文本身。

3.4.2 故障注入目标识别。基于选定的跟踪（第 3.3 节）和故障库，下一步是确定故障注入的位置，即故障注入目标。为实现全面的故障覆盖率，我们为注入目标引入了一种称为端点的细粒度抽象。端点是一个元组（组件、框架、方法）：组件定义交互的语义上下文（如数据库、缓存）；框架标识具体的库或实现（如 MyBatis、Jedis）；方法指定操作（如更新、获取）。这种抽象使我们能够区分由不同团队通过不同框架实现的功能相同的操作，而每种实现都具有独特的故障处理特性。例如，在用户查询中，我们可以区分 RPC-gRPC-按 ID 查找用户与 RPC-Dubbo-按 ID 查找用户这类关键端点对；在消息传递操作中，可以区分 MQ-Kafka-发送与



（a）生产者-消费者数据依赖模式



（b）双写数据依赖模式

图 3：数据流依赖模式

MQ-Pulsar-发送。这种精确度对于系统地发现特定实现的漏洞至关重要。每个端点都可作为目标，从我们的故障库中选取相关故障进行注入，从而针对单个操作测试各种故障。例如，可以向 MQ-Kafka-发送端点注入超时异常以测试网络问题，或注入 SerializationException 以模拟数据格式不匹配。

从技术上讲，跟踪中的每个跨度都对应一个端点，因为它记录了微服务系统内部执行的一项操作（第 3.2 节）。然而，由于单个复杂跟踪可能包含数百甚至数千个跨度，向其所有端点注入故障会导致潜在测试用例的组合爆炸。为解决这一问题，Cast 采用了一种有原则的剪枝机制，通过以下三种策略系统地缩减测试空间，剔除冗余测试并优先安排高价值测试：

- 跨服务采样：给定端点（例如数据库调用）的故障处理逻辑通常在共享库或框架中实现，因此其行为在不同服务之间保持一致。对每个调用该端点的微服务都进行全面测试，效率很低。因此，我们随机抽取少量可配置数量的不同微服务（例如三个）进行故障注入。该方法承认，即便使用相同的框架，不同实现中的弹性问题仍可能表现各异。

- 跟踪内冗余消除：单条跟踪中可能包含同一端点的多个跨度（例如，重复的数据库查询）。为避免冗余，我们只在最后一次调用时注入故障。理由是，最后一次调用是在

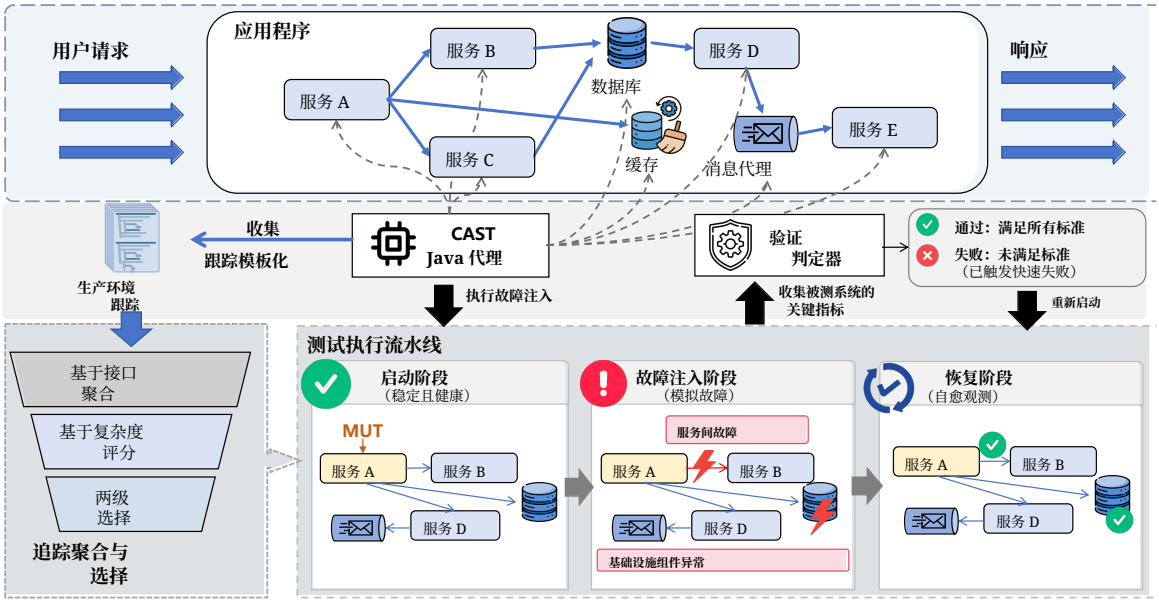


图 4：Cast 系统的测试执行流水线

更高级的系统状态下执行的，该状态受此前所有操作的影响，因此更有可能暴露复杂的、依赖于状态的漏洞。此外，对于数据库事务，在最后一次调用时注入故障，可以测试系统回滚完全暂存事务的能力；与回滚较早的部分操作相比，这通常更容易出错。

- 数据流依赖优先级排序：该策略通过分析跟踪中的数据流依赖结构，对注入目标进行优先级排序，因为这些依赖容易导致级联故障。根据载荷和操作序列中的数据重叠情况，我们识别出两种基本的依赖结构，如图 3 所示。第一种模式，生产者-消费者 （图 3a），涉及某个服务（例如 OrderService）先调用一个组件获取数据（例如调用 GeoService 验证地址），然后在后续调用另一个组件时使用所获取的数据（例如使用经过验证的地址调用 ShippingService）。整个工作流的完整性取决于数据能否从生产者（GeoService）成功流向消费者（ShippingService）。我们将生产者列为高价值注入目标，因为它发生故障时可能会沿数据链传播，使我们无需在同一跟踪中单独测试下游消费者。第二种模式，双写 （图 3b），指服务中的同一组数据（例如 ProductService 中的产品 ID 和价格）被用于调用两个或多个独立操作（例如在数据库中执行 UPDATE，在缓存中执行 DELETE）。这种模式常见于数据同步任务，并且极易引发隐蔽但严重的不一致错误。尤其是出于性能考虑，对缓存的第二次写入通常以异步方式执行，此时服务不会等待确认。在这种情况下，我们跳过主要的同步写入，专门关注次要的异步操作。由于主要写入通常已由生产者-消费者测试或单个服务测试覆盖，因此，关注次要写入可以锁定造成隐蔽数据不一致的特定源头，同时显著减少所需测试用例的数量。如第 3.5 节所述，这种场景需要更复杂的验证预言机。

这种有原则的剪枝机制能够生成一组具有策略多样性的故障注入测试用例，覆盖微服务中广泛的操作和框架。它使我们能够在无需承担穷举测试成本的情况下，验证常见的故障处理模式，从而兼顾覆盖率与效率。

3.5 自动化测试执行与验证

利用这些故障注入测试用例，最后一个阶段将系统地执行这些用例，并验证系统的韧性。

3.5.1 执行流水线与测试调度。 每个测试用例都在定义明确的三阶段周期内执行。图 4 从运行时视角展示了这一过程，详细说明 Java 代理在各阶段如何与被测微服务（MUT）及平台组件交互。具体而言，每个阶段都配置了用户定义的持续时间和注入频率，以确保系统有充足的时间趋于稳定，并使相关影响得以观察。在这一过程中，我们通过重放选定的跟踪，持续向 MUT 生成流量。

- 启动阶段：MUT 以独立的按需容器形式部署。为确保严格的数据隔离并避免污染共享环境，我们为每个测试初始化一个全新的实例。在此阶段不注入任何故障，以确保系统处于稳定、健康的状态。
- 故障注入阶段：选定的故障被激活，针对指定的端点。这使我们能够在特定故障条件下监控系统的行为。

- 恢复阶段：故障注入被停用。这一最后阶段对于观察系统能否自主恢复并回到其原有的健康运行状态至关重要。

这一过程中的一个实际挑战是被测系统启动所带来的开销。此过程可能耗时较长，因为它涉及应用程序和环境的初始化，以及服务达到稳定运行状态所需的时间。每次启动只执行一次故障注入会慢得难以接受。为摊销这项开销，Cast 将多个故障注入测试分组到一次运行中。我们采用贪心调度算法，迭代选择能够最大程度提升新增端点覆盖率的跟踪。这样可以最大限度地减少执行所有故障注入测试所需的启动次数。在单次测试运行中，这些批处理测试会按顺序执行。

然而，这种设计可能会导致状态污染，即前一个测试引发的故障改变系统状态，进而影响后续测试。为缓解这一问题，Cast 采用“快速失败”执行模型。具体而言，只要每个测试通过，流水线就会继续执行下一个测试。一旦测试失败，表明可能存在漏洞，整个运行就会立即停止。随后，系统会完全重新启动，以确保剩余测试从干净状态开始。我们承认，即使之前通过的测试也可能悄然改变系统状态。不过，我们认为这是为显著提升效率而做出的务实权衡。实际上，这类情况很少见，而且我们的跨服务采样策略本身具有冗余性。此外，为提升多个测试周期中的长期有效性，Cast 会保留已执行测试用例的历史记录。根据后续批次的录制流量安排新测试时，会利用这些历史记录避免重新执行完全相同的测试。

3.5.2 验证预言机。为了自动确定测试结果，Cast 采用了一个多方面的验证预言机，包含以下两个关键组件：

- 基于阶段的性能标准：我们的判定器定义了一组性能标准，即关于被测系统关键指标（例如成功率、延迟和吞吐量）的可量化假设。具体的标准值源自历史跟踪数据，并且可能因执行阶段而异。例如，启动阶段要求成功率达到 100%，以确认系统初始化正确。为了验证故障确实产生了影响，预计故障注入阶段的成功率会显著下降（例如，$\leq 30\%$）。在恢复阶段，成功率必须恢复到健康水平（例如，$\geq 80\%$），以证明系统具备自我修复能力。该阈值有意设置为低于 100%，以便在系统稳定过程中为现实的恢复时间窗口留出余地。尽管这种自动化与绝对精确度之间的权衡偶尔可能导致假阳性或假阴性（见第 4.2 节），但它能够实现实用的自动化验证。

- 细粒度断言点：只检查服务入口点的最终 HTTP 响应代码，是一种过于简单的方法，往往不足以完成验证。我们发现，对于某些异步操作，例如向 Pulsar 发布消息，即使 Pulsar 随后未能将消息送达消费者，调用也可能成功返回应用程序逻辑。这种“即发即弃”行为中，服务会在任务提交后便视为完成，而不等待其实际送达，这会导致错误的通过判定。会导致错误的通过判定。为解决这一问题，Cast 不仅在服务入口点，还直接在注入故障的内部端点设置断言点。通过这种双层验证，我们可以准确评估故障的直接影响，以及它对服务整体行为的影响。

只有当观察到的行为在所有相关断言点上均满足全部三个阶段所定义的标准时，测试才被视为通过。失败的测试会被标记，以供进一步调查。

4 评估

在本节中，我们评估 Cast 在生产环境部署中的表现。具体而言，我们旨在回答以下两个研究问题：

- **RQ1（可重放性）**：Cast 在使录制流量可重放方面有多有效?
- **RQ2（有效性）**：Cast 在自动发现韧性漏洞方面有多有效?

Cast 作为集中式韧性测试平台部署于华为云。八个多月以来，许多开发团队使用 Cast 主动识别其应用程序中的韧性问题。鉴于 Cast 是一个共享平台，各团队可独立运行测试并分析结果，因此手动跟踪并确认所有服务团队报告的每一项潜在漏洞的结果并不现实。因此，在本文中，我们将分析重点放在四个具有代表性的应用程序（匿名化为服务 1-4）的测试结果上。选择它们的具体原因是，其特征与 Cast 旨在解决的挑战相一致，即大规模（数百个微服务）、大量业务关键流量，以及具有错综复杂依赖关系的架构复杂性。

4.1 RQ1：录制流量的可重放性

为回答 RQ1，我们评估 Cast 的动态变量识别技术在使生产流量可重放方面的有效性。成功重放是指重放的请求被目标服务接受并返回预期的成功状态码（例如，HTTP 2xx），从而在进行故障注入前建立基线。

我们首先从目标应用程序收集六小时的流量（超过三百万条跟踪）。Cast 的聚合模块将这些数据精简为 291 个独立的服务接口调用。对于每个接口，Cast 会分析其所有关联的跟踪实例，并应用两阶段启发式方法（第 3.2 节），自动识别并参数化对时间敏感的变量和状态依赖变量。为验证结果，我们从 291 个接口调用中各随机选取一个跟踪实例，并尝试在实时测试系统上重放。在 291 次重放尝试中，Cast 成功重放了 286 条跟踪；使用我们的模板化技术，成功率从低于 80%（朴素重放）提升至 98.3%。这一高成功率表明，我们的自动化方法能够有效解决绝大多数接口的状态依赖问题。使用不同随机样本进行的重复实验结果一致，证实了该方法的稳定性。

我们对5个失败案例进行了人工调查，发现它们主要是由高度依赖具体业务或系统的

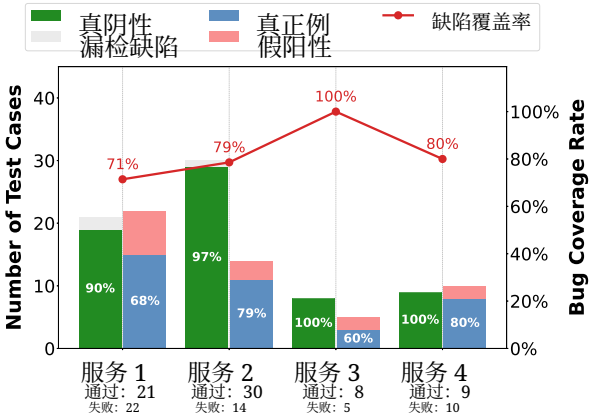


图5：不同服务的实验结果

变量导致的，这些变量无法通过我们的通用识别方法识别。例如，一些请求包含动态计算的安全签名，或由复杂的专有业务逻辑生成、且未体现在请求-响应负载结构中的令牌。为在实际工业环境中处理这些边缘情况，Cast 维护了一个可配置列表，允许工程师手动为特定服务注册或注销变量。这种混合方法只需针对这些特殊情况进行一次少量人工操作，即可实现全面的重放覆盖率。

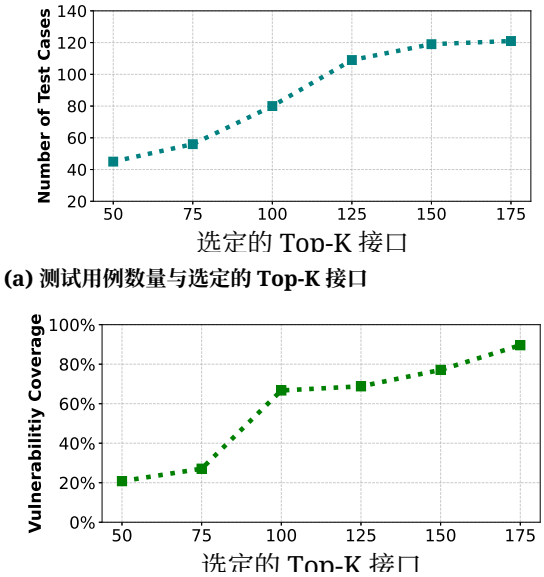
4.2 RQ2：韧性缺陷检测的有效性

为评估Cast的有效性，我们首先考察其对真实生产系统的长期影响。在华为云为期八个月的部署期间，不同团队持续采用Cast来提升服务可靠性。本文展示了Cast在四个大规模且具有代表性的应用程序中的使用结果。总体而言，Cast成功发现了137个潜在漏洞；截至本文撰写时，其中89个已通过开发人员的人工调查得到确认。这体现了Cast在生产环境中的实际价值。

为了对其检测能力进行更可控的定量分析，我们收集了一个包含48个已确认漏洞的真值数据集。这些案例包括通过事故报告和人工调查发现的历史缺陷，为我们的实验提供了全面的基准。我们使用与RQ1相同的六小时录制流量语料库，针对包含这些缺陷的系统生成测试用例。按照我们的复杂度驱动选择策略（第3.3节），我们选取复杂度分数最高的150个接口及其代表性追踪。随后，我们应用剪枝逻辑（第3.4节）来识别端点。对于每种端点，我们随机选择至多三个不同的微服务进行测试。此过程生成了119个不同的测试用例，覆盖477个故障注入目标。

图 5 的详细结果显示，在执行 119 个测试用例后，68 个通过，51 个失败。Cast 的真阴性率很高，这意味着几乎所有通过的测试用例确实都不存在漏洞。对 51 个失败测试的人工调查发现

Cast 成功识别了 48 个已知漏洞中的 37 个，达到了



(a) 测试用例数量与选定的 Top-K 接口

(b) 漏洞覆盖率与选定的 Top-K 接口

图 6：不同 Top-K 接口下的敏感性分析

在不同服务中始终保持较高的缺陷覆盖率（总体为 77.1%）。其余 14 个失败的测试被判定为假阳性。在这些情况下，故障被正确注入，系统行为也发生了变化，但这并不代表真正的韧性缺陷。详细分析表明，其中 9 个假阳性是由于默认性能阈值对某些非关键后台操作（例如指标收集、日志记录）过于敏感，另外 5 个则是故障条件下合理的、虽有降级但仍可接受的行为。同样，对于 3 个假阴性，测试用例虽已执行，但判定器错误地报告“通过”。这通常发生在故障影响过于轻微、不足以触发默认性能阈值时，例如成功率降至 35%（略高于 30% 的阈值），却仍代表显著的性能下降。这些结果凸显了自动化预言机设计中敏感性与特异性之间固有的权衡。为此，Cast 允许开发人员配置自定义阈值，从而逐步提高判定器的精确度。

我们还分析了 Cast 未能检测到的 11 个漏洞。除 3 个假阴性外，其余 8 个缺陷均源于测试生成局限性：从未针对易受攻击路径创建测试。具体而言，其中 5 个缺陷未能被发现，是因为录制的流量语料库中不包含能够触发易受攻击路径的请求，这是任何基于录制与重放的方法都存在的根本局限。另有 3 个缺陷与复杂度分数不在 Top-K 选择范围内的接口有关。为缓解这一问题，Cast 采用了历史执行感知选择机制。该机制会保留以往已执行测试的历史记录。生成新的测试套件时，它会优先对尚未测试或此前失败的接口进行故障注入，从而有效避免重复执行已通过的测试。系统会定期重置这一历史记录（例如每周一次），以确保重新验证系统是否存在潜在回归问题，在探索新的测试空间与持续验证之间取得平衡。

4.3 参数敏感性分析

为进一步研究测试套件规模与缺陷覆盖率之间的权衡，我们通过将 Top-K 接口的数量从 50 调整到 175，进行敏感性分析，如图 6 所示。我们首先考察执行成本，该成本以生成的测试用例数量衡量（图 6a）。随着最初的接口逐步加入，测试套件规模显著增长，因为这些接口引入了许多独特的端点。然而，随着 K 增大，曲线开始趋于平缓，因为新增跟踪中很可能包含先前跟踪已经覆盖的端点。这表明我们的剪枝逻辑能够有效消除冗余测试用例。接着，图 6b 展示了相应的漏洞覆盖率。曲线在 K=75 和 K=100之间出现一个关键拐点，覆盖率大幅提升至近 70%。这验证了我们的端点覆盖率引导策略能够凭借足够的跟踪找到关键故障注入目标。与图 6b 类似，这条曲线也显示出收益递减：每增加一条跟踪，对总体覆盖率的贡献都会逐渐降低。

这两条曲线凸显了覆盖率与成本之间的根本权衡。较大的 K 虽然能在单次运行中实现更高的覆盖率，但需要执行更多测试用例，这对于频繁的测试周期而言可能并不实际。这正是 Cast 的历史执行感知机制发挥关键作用之处。团队无需依赖单次穷尽式运行，而可以在日常或持续测试中采用较小的 K（例如 100）。随着多次运行的进行，Cast 的历史跟踪可确保其避免重新测试此前已通过的场景，并逐步探索其他接口。这一策略使 Cast 能够随着时间推移，以较少的每日执行量实现较高的累计覆盖率。

4.4 案例研究

为了进一步说明 Cast 发现的错误类型，并展示其设计原则的有效性，我们介绍部署过程中两个具有代表性的案例研究。

- **案例 1：未处理 HTTP 超时导致的级联故障**。某个微服务——服务 A——负责处理上传的数据。其工作流程包括接收请求，通过 HTTP REST API 查询第三方服务，以验证并补充传入的信息，然后将最终消息发布到 Pulsar 队列。当 Cast 向此 HTTP 调用注入 java.net.SocketTimeoutException 时，由于缺少超时策略，工作线程会无限期挂起。这导致初始请求失败，并使该线程无法处理任何后续请求，进而造成该业务功能彻底且持续地失效。根本原因是，用于调用第三方服务的 RestTemplate HTTP客户端在实例化时未设置读取超时。这凸显了 Cast 的数据流依赖优先级排序的价值。该工作流程体现了经典的生产者-消费者模式（用户 → 服务 A → 3第三方服务 → Pulsar），而 Cast 的复杂度驱动选择会优先选择此类多跳跟踪，因为它们更有可能导致级联故障。

- **案例 2：异步通信中的静默失败**。负责添加新数据条目的服务会先将数据写入数据库，然后向 Kafka 主题发布通知事件。当 Cast 在 Kafka 发布过程中注入 DisconnectException 时，该服务未能捕获此错误。因此，尽管内部操作失败，原始 REST API 调用仍会向调用方返回 HTTP 200 OK 成功状态码。数据已正确持久化，但下游服务始终未收到通知，导致整个系统出现数据不一致。根本原因在于采用了“即发即弃”模式：应用程序触发异步 Kafka 发布操作后，却未检查其返回值或处理异常。这一漏洞凸显了 Cast 的细粒度断言点至关重要。仅检查最终 HTTP响应的简单判定器会漏掉这个错误。通过直接在内部 Kafka 生产者端点设置断言点，Cast 能正确识别故障，揭示这一隐蔽但严重的问题。

4.5 效度威胁

外部效度。 我们的评估仅在华为云上进行，这可能会限制研究结果的普适性。不过，我们认为，研究发现具有广泛的适用性，原因如下。首先，我们关注的架构模式、平台组件（Kafka、Redis、MySQL）和通信协议（HTTP、gRPC）在业界得到广泛采用。其次，我们的设计原则，即黑盒流量分析、应用级故障注入和多方面测试预言，与平台无关。最后，我们所应对的挑战（状态依赖、测试空间爆炸、测试预言设计）是韧性测试中的根本问题，与具体的云平台无关。

内部效度。 有几个因素可能会影响结果的准确性。首先，依赖基于阈值的测试预言可能导致假阳性或假阴性。为缓解这一问题，Cast 允许团队根据自身需求自定义阈值，并根据运行反馈持续改进阈值（第 3.5 节）。其次，我们的复杂度驱动选择是一种启发式方法，可能会遗漏较简单接口中的错误。为此，我们采用历史执行感知机制，在多次运行中逐步探索尚未测试的接口，从而实现累积覆盖。第三，录制回放方法仅限于捕获流量中出现的执行路径。尽管这是一个根本性限制，我们仍通过持续记录新的流量模式，并维护一个随时间不断扩充、日益多样化的轨迹语料库来缓解这一问题，从而捕获越来越全面的执行场景。

5 条经验教训

在工业环境中设计、部署和评估 Cast，让我们对微服务系统韧性测试的本质有了若干重要认识。我们将这些经验提炼为以下几条关键教训。

验证预言必须超越 API 边界。在研究初期，我们曾考虑一种更简单的判定器模型，只检查最终面向用户的 API 响应（例如 HTTP 200 OK），这也是现有研究中的常见做法。但事实证明，这种方法并不充分。我们反复遇到这样的情况，尤其是在异步操作中：主要 API 调用成功了，但关键的后台操作却悄无声息地失败了。正如我们的案例研究（第 4.4 节）所示，这可能导致严重的数据不一致，而简单的判定器无法发现这些问题。这一经历凸显了一个根本性的教训：有效的韧性测试需要深入了解系统内部。值得信赖的验证预言必须设置

断言点，不仅要覆盖服务边界，还要覆盖关键的内部端点，例如平台组件交互，以捕获那些不会立即传递给用户的故障。

通过自动化实现的累积覆盖率胜过完美的优先级排序。尽管许多研究 [16, 17, 37, 38] 采用复杂策略来确定故障注入点的优先级，但我们发现，在复杂、由多个团队协作的工业环境中，预测缺陷位置颇具挑战。我们发现，测试系统的实际价值可以近似表示为：价值 = （每次运行的覆盖率 × 频率）/投入。需要大量人工投入或复杂系统模型的“完美”优先级排序策略，可能实现较高的单次运行覆盖率，但由于运营开销，运行频率较低。相比之下，我们“足够好”的复杂度驱动启发式方法结合全面自动化，可支持每日或每周进行测试运行。关键在于设计一个具备历史感知能力的系统，以智能地避免重新测试已通过的场景。这样，框架便能随着时间推移不断累积成效，在后续测试运行中系统地探索新的、优先级较低的接口。这种迭代方法能够随着时间推移实现较高的累积覆盖率，同时每天只需执行规模更小、更易管理的测试，从而为提升系统可靠性提供一条更实用、更稳健的路径。

工业系统中的工程权衡不可避免。学术原型在简单应用程序上往往前景可观，但工业系统的复杂性和规模带来了挑战，需要采取不同的工程权衡。在 Cast 的开发过程中，我们面临着许多设计决策：理论上的最优性与实际约束相互冲突。例如，我们的基于载荷的变量识别启发式方法不如静态污点分析精确，但无需访问源代码即可扩展到数百项服务。我们固定的判定阈值偶尔会导致假阳性，但能够实现完全自动化，无需针对每项测试进行人工配置。同样，我们的复杂度指标比基于图的算法更简单，但不需要系统模型，也无需维护依赖关系。这些权衡反映了工业软件工程的一个基本现实：能够发现 80% 缺陷的已部署系统，比仍停留在研究实验室中的完美系统更有价值。关键在于有意识地进行这些权衡，为有特定需求的团队提供配置选项，并根据运行反馈持续迭代。

6 相关工作

6.1 弹性测试与混沌工程

通过有意注入故障进行韧性测试，已从粗粒度的基础设施故障发展到应用层的复杂测试。早期方法（如 Netflix 的 Chaos Monkey [13, 14] ）开创了随机终止虚拟机以测试系统恢复的实践。虽然这些方法对基础设施鲁棒性测试有效，但无法触发应用程序特有的错误处理逻辑 [27, 40]。这一局限推动了更细粒度方法的发展。例如，Gremlin [27] 会拦截服务间消息；Filibuster [40] 将静态分析与混合具体符号执行结合起来，用于 HTTP 服务；而 Doctor [1], 、Setsudo [32], 和 FATE/DESTINI [25]则注入较低层级的 I/O 故障。

弹性测试的核心挑战在于管理故障空间的组合爆炸 [37, 52]。复杂的优先级排序策略应运而生，以应对这一问题。例如，LDFI [5] 从正确执行结果反向推理，以识别最小故障集；IntelliFT [37] 利用适应度引导反馈，朝着具有影响力的场景进行引导；MicroFI [16] 则采用 PageRank 算法进行优先级排序。尽管这些方法在理论上十分优雅，但在工业环境中的采用仍面临重大障碍。LDFI 要求使用确定性执行模型，这对于多团队环境并不切实际；适应度引导方法需要多次迭代，不适用于大规模系统；而基于图的方法则难以应对动态服务拓扑。近期的综述 [44] 和量子机器学习优化 [12] 进一步凸显了复杂性与效率之间的权衡。Cast 通过一种由复杂性驱动的启发式方法应对这些实际限制：该方法无需系统模型，并可在单次遍历中收敛——正如我们为期八个月的部署所验证的那样，它以理论最优性换取了运营可行性 [4, 42, 47, 56]。

测试框架的发展反映了工业界的需求。具体而言，CHESS [39] 面向自适应系统；自适应混沌工程采用强化学习；Rainmaker [18]为云应用提供一键式测试，而 MicroRes [54] 则分析退化模式。近期的部署揭示了实用层面的见解。例如，Chen 等人 [19]展示了通过大规模故障注入，云边环境中的响应稳定性提升了 80%。预言机问题仍然至关重要。基于大语言模型的方法（如 CANDOR [53] ）通过共识生成测试预言，SATORI [3] 则从 API 规范 [24, 31],中推导测试预言；Cast 采用实用的基于阈值的测试预言，并提供可配置参数，无需针对每项测试进行手动配置，即可实现全自动化。

6.2 流量录制与回放和模拟

通过流量录制与回放创建逼真的测试场景，对于实现高保真测试已变得至关重要 [10, 35]。其根本挑战在于状态依赖，即生产流量包含会导致回放失败的时间敏感变量。WaRR [6], Mugshot [41], 和 WebRR [36]等 Web 应用工具通过各种浏览器特定机制解决了这一问题。在微服务中，由于分布式状态分散在各个服务之间，这一挑战更加突出。Liu 等人 [35] 使用静态污点分析进行变量识别，但要求与 SOFA/SOFABoot 框架耦合；而 Zhu 等人 [59] 则利用人工智能提供模拟点推荐 [8, 9]。

模拟传统上用于单元测试 [45, 46], , 如今正越来越多地融入系统级韧性测试。EvoMaster [57]支持测试种子注入和服务模拟，而 MicroFuzz [21] 则明确针对微服务环境复杂性。Cast 的独特之处在于采用黑盒、以负载为中心的方法，将服务视为不透明实体。我们的两阶段启发式方法，即识别同时出现在请求和响应中的令牌，再依据跨跨度变异性进行筛选，在无需访问源代码或依赖框架的情况下，实现了 98.3% 的回放成功率。尽管这种设计选择的精确度可能不如白盒分析 [34],, 但对于团队使用多种语言和框架的多语言工业环境而言，它至关重要。与需要大量系统知识或训练数据的方法不同，Cast 通过轻量级动态分析即可立即发挥价值，因此适用于持续演进的生产系统。

7 结论

本文介绍 Cast，这一框架旨在应对生产微服务系统中韧性测试的关键挑战。Cast 的核心贡献在于其整体方法，可用于创建真实、可扩展且完全自动化的韧性测试。它将源自实时流量的高保真工作负载与细粒度的应用级故障库相结合，弥合测试与生产实际之间的差距。为应对组合复杂性，Cast 采用复杂度驱动的方法，缩减庞大的测试空间，从而能够高效而全面地评估最关键的执行路径。Cast 已在华为云部署八个多月，成功发现 137 个潜在韧性漏洞，其中 89 个已由开发人员确认。此外，我们还针对包含 48 个已复现缺陷的基准测试集开展了受控实验，Cast 的缺陷覆盖率达到 90%。这些结果证实，我们的方法不仅实现了生产流量的高可重放性，还能高效发现现实世界中的关键漏洞。

致谢

本工作得到国家自然科学基金委员会（编号：62402536）的支持。

参考文献

[1] 1995. DOCTOR：面向分布式实时系统的集成式软件故障注入环境。载于《国际计算机性能与可靠性研讨会论文集》（*IPDS '95*）。IEEE计算机学会，美国，204。[2] 阿里巴巴。2025年。JVM Sandbox。检索于2025年12月，网址：https://github. com/alibaba/JVM-Sandbox [3] Juan C. Alonso、Alberto Martin-Lopez、Sergio Segura、Gabriele Bavota 和 An- tonio Ruiz-Cortés。2025年。SATORI：REST API静态测试预言生成。*CoRR* abs/2508.16318（2025年）。arXiv:2508.16318 doi:10.48550/ARXIV.2508.16318[4] Peter Alvaro、Kolton Andrus、Chris Sanden、Casey Rosenthal、Ali Basiri 和 Lorin Hochstein。2016年。互联网规模的自动化故障测试研究。载于《第七届*ACM*云计算研讨会论文集》，美国加利福尼亚州圣克拉拉，*2016*年*10*月*5*日至*7*日，Marcos K. Aguilera、Brian Cooper 和 Yanlei Diao（编）。ACM，17 28 10 1145 2987550 2987555 – 。doi: . / . Peter Alvaro、Joshua Rosen 和 Joseph M. Hellerstein。2015年。数据血缘驱动的故障注入。载于《*2015*年*ACM SIGMOD*数据管理国际会议论文集》，澳大利亚维多利亚州墨尔本，*2015*年*5*月*31*日至*6*月*4*日，Timos K. Sellis、Susan B. Davidson 和 Zachary G. Ives（编）。ACM，331–346。doi:10.1145/2723372.2723711[6] Silviu Andrica 和 George Candea。2011年。WaRR：用于高保真Web应用记录与回放的工具。载于《*2011*年*IEEE/IFIP*可信系统与网络国际会议论文集》，*DSN 2011*，中国香港，*6*月 *27 30 2011* 403 410 10 1109 2011 5958253 -。IEEE计算机学会，– 。doi: . /DSN. 2019 。Andrea Arcuri。 。使用 EvoMas- ter 生成RESTful API自动化测试用例。*ACM*软件工程与方法学汇刊 28, 1 2019 3 1 3 37 10 1145 3293455 ( ), ： –: 。doi: . / Andrea Arcuri、Gordon Fraser 和 Juan Pablo Galeotti。 。为具有环境依赖的类自动生成单元测试。载于*ACM/IEEE*自动化软件工程国际会议，*ASE '14*，瑞典韦斯特罗斯，*2014*年*9*月*15*日至*19*日，Ivica Crnkovic、Marsha Chechik 和 Paul Grünbacher（编）。ACM，79 90 10 1145 2642937 2642986 – 。doi: . / . Andrea Arcuri、Gordon Fraser 和 Juan Pablo Galeotti。2015年。为自动化单元测试生成TCP/UDP网络数据。载于《*2015*年第*10*届软件工程基础联合会议论文集》，*ESEC/FSE 2015*，意大利贝加莫，*2015*年*8*月*30*日至*9*月*4*日，Elisabetta Di Nitto、Mark Harman 和 Patrick Heymans（编）。ACM，155 165 10 1145 2786805 2786828 – 。doi: . / . Nipun Arora、Jonathan Bell、Franjo Ivancic、Gail E. Kaiser 和 Baishakhi Ray。2018年。面向服务的应用中无需录制即可重放生产环境缺陷。载于《第*33*届*ACM/IEEE*自动化软件工程国际会议论文集》，*ASE 2018*，法国蒙彼利埃，*2018*年*9*月*3*日至*7*日，Marianne Huchard、Christian Kästner 和 Gordon Fraser（编）。ACM，452–463。doi:10.1145/3238147. 3238186[11] Vaggelis Atlidakis、Patrice Godefroid 和 Marina Polishchuk。2019年。RESTler：有状态REST API模糊测试。载于《第*41*届软件工程国际会议论文集》，*ICSE 2019*，加拿大魁北克省蒙特利尔，*2019*年*5*月*25*日至*31*日，Joanne M。Atlee、Tevfik Bultan和Jon Whittle（编）。IEEE / ACM，748–758。doi:10.1109/ICSE.2019.00083[12] Gopichand Bandarupalli。2025年。量子优化与机器学习相结合的软件测试之影响。*CoRR* abs/2506.02090（2025年）。arXiv:2506.02090 doi:10.48550/ARXIV.2506.02090[13] Ali Basiri、Niosha Behnam、Ruud de Rooij、Lorin Hochstein、Luke Kosewski、Justin Reynolds和Casey Rosenthal。2016 33, 3 2016 35 41 10 1109 2016 。混沌工程。*IEEE*软件（ ），– 。doi: 。 /MS. 。 60 [14] 2012 Cory Bennett和Ariel Tseitlin。。Chaos Monkey被释放到真实环境。*Netflix* 技术博客 30, 1（2012）。[15] Matteo Camilli、Antonio Guerriero、Andrea Janes、Barbara Russo和Stefano Russo。2022。微服务集成性能与可靠性测试。见*IEEE/ACM*国际软件测试自动化会议，*AST@ICSE 2022*，美国宾夕法尼亚州匹兹堡，*2022*年*5*月*21-22*日。ACM/IEEE，29–39。doi:10.1145/3524481. 3527233 [16] Hongyang Chen、Pengfei Chen、Guangba Yu、Xiaoyun Li和Zilong He。2024。MicroFI：面向微服务应用的非侵入式优先请求级故障注入。*IEEE*可靠与安全计算汇刊 21, 5（2024），4921–4938。doi:10.1109/TDSC.2024.3363902[17] Haicheng Chen、Wensheng Dou、Dong Wang和Feng Qin。2020。CoFI：面向云系统的一致性引导的故障注入。见 第*35*届*IEEE/ACM*自动化软件工程国际会议，*ASE 2020*，澳大利亚墨尔本，*9*月 *21 25, 2020* 536 547 10 1145 3324884 3416548 -。IEEE，– 。 [18] 2023 doi: 。 / 。 Yinfang Chen、Xudong Sun、Suman Nath、Ze Yang和Tianyin Xu。。使用Rainmaker对云后端应用进行一键可靠性测试。见第*20*届*USENIX*网络系统设计与实现研讨会，*NSDI 2023*，马萨诸塞州波士顿，*2023*年*4*月*17-19*日，Mahesh Balakrishnan和Manya Ghobadi（编）。USENIX协会，1701–1716。https://www.usenix.org/conference/ nsdi23/presentation/chen-yinfang [19] Zihao Chen、Mohammad Goudarzi和Adel Nadjaran Toosi。2025年。通过故障注入对云边环境中Kubernetes进行韧性评估。*CoRR* abs/2507.16109（2025年）。arXiv:2507.16109 doi:10.48550/ARXIV.2507.16109[20] Zhuangbin Chen、Yu Kang、Liqun Li、Xu Zhang、Hongyu Zhang、Hui Xu、Yangfan Zhou、Li Yang、Jeffrey Sun、Zhangwei Xu、Yingnong Dang、Feng Gao、Pu Zhao、Bo Qiao、Qingwei Lin、Dongmei Zhang和Michael R. Lyu。2020。迈向智能事件管理：我们为何需要它以及如何实现。见 *ESEC/FSE '20*：第*28*届*ACM*欧洲软件工程联合会议暨软件工程基础研讨会，美国线上活动，*2020*年*11*月*8-13*日，Prem Devanbu、Myra B. Cohen和Thomas Zimmermann（编）。ACM，1487–1497。doi:10.1145/3368089.3417055[21] Peng Di、Bingchang Liu和Yiyi Gao。2024。MicroFuzz：一种面向微服务的高效模糊测试框架。见 第*46*届国际软件工程大会论文集：软件工程实践，*ICSE-SEIP 2024*，葡萄牙里斯本，*4*月 *14 20, 2024* 216 227 10 1145 3639477 3639723 -。ACM，– 。doi: 。 [22] 2021 / 。 FudanSELab。。Train Ticket—— 一个基准微服务系统。于 2025 年 7 月检索自 https://github.com/FudanSELab/train-ticket [23] Luca Giamattei、Antonio Guerriero、Roberto Pietrantuono和Stefano Russo。2022。微服务架构的自动化灰盒测试。见 第*22*届*IEEE*软件质量、可靠性与安全国际会议，*QRS 2022*，中国广州，*2022*年*12*月*5-9*日。IEEE，640–650。doi:10.1109/QRS57517. 2022.00070[24] Suavis Giramata、Madhusudan Srinivasan、Venkat Naidu Gudivada和Upulee Kanewala。2025年。大语言模型中的高效公平性测试：为偏差检测优先排序蜕变关系。*CoRR* abs/2505.07870（2025年）。arXiv:2505.07870 doi:10.48550/ARXIV.2505.07870[25] Haryadi S. Gunawi、Thanh Do、Pallavi Joshi、Peter Alvaro、Joseph M. Hellerstein、Andrea C. Arpaci-Dusseau、Remzi H. Arpaci-Dusseau、Koushik Sen和Dhruba Borthakur。2011。FATE与DESTINI：一种云恢复测试框架。见 第*8*届*USENIX*网络系统设计与实现研讨会论文集，*NSDI 2011*，美国马萨诸塞州波士顿，*2011*年*3*月*30*日-*4*月*1*日，David G. Andersen和Sylvia Ratnasamy（编）。USENIX协会。https://www.usenix.org/conference/nsdi11/fate-and-destini-framework-cloud-recovery-testing [26] Pinjia He、Jieming Zhu、Zibin Zheng和Michael R. Lyu。2017。Drain：一种采用固定深度树的在线日志解析方法。见 *2017*年*IEEE Web*服务国际会议，*ICWS 2017*，美国夏威夷州檀香山，*2017*年*6*月*25-30*日，Ilkay Altintas和Shiping Chen（编）。IEEE，33 40 10 1109 2017 13 – 。doi: 。 /IC [27] WS. 。 Victor Heorhiadi、Shriram Rajagopalan、Hani Jamjoom、Michael K. Reiter和Vyas Sekar。2016。Gremlin：微服务的系统化韧性测试。见第*36*届*IEEE*分布式计算系统国际会议，*ICDCS 2016*，日本奈良，*2016*年*6*月*27-30*日。IEEE计算机学会，57–66。doi:10.1109/ICDCS. 2016.11[28] Peng Huang、Chuanxiong Guo、Jacob R. Lorch、Lidong Zhou和Yingnong Dang。2018。捕获并增强在位系统可观测性以进行故障检测。见第*13*届*USENIX*操作系统设计与实现研讨会，*OSDI 2018*，美国加利福尼亚州卡尔斯巴德，*2018*年*10*月*8-10*日，Andrea C. Arpaci-Dusseau和Geoff Voelker（编）。USENIX协会，1–16。https://www.usenix.org/ conference/osdi18/presentation/huang [29] Peng Huang、Chuanxiong Guo、Lidong Zhou、Jacob R. Lorch、Yingnong Dang、Murali Chintalapati和Randolph Yao。2017。灰色故障：阿喀琉斯之踵

云规模系统的。见 第16届操作系统热点主题研讨会论文集，*HotOS 2017*，加拿大不列颠哥伦比亚省惠斯勒，*2017*年*5*月*8-10*日，Alexandra Fedorova、Andrew Warfield、Ivan Beschastnikh和Rachit Agarwal（编）。ACM，150–155。doi:10.1145/3102980.3103005[30] Mingxuan Hui、Lu Wang、Hao Li、Ren Yang、Yuxin Song、Huiying Zhuang、Di Cui和Qingshan Li。2025年。揭示微服务测试方法、挑战、解决方案及解决方案差距：一项系统映射研究。*J. Syst. Softw.* 220（ 2025 112232 10 1016 2024 112232 ），。doi:。/J.JSS.。Shan Jiang、[31] Chenguang Zhu和Sarfraz Khurshid。2024。使用大语言模型生成可执行预言，以检查客户端代码是否符合JDK Javadoc文档的要求。*CoRR* abs/2411.01789（2024）。arXiv:2411.01789 doi:10.48550/ARXIV. 2411.01789[32] Pallavi Joshi、Malay K. Ganai、Gogul Balakrishnan、Aarti Gupta和Nadia Pa- paponstantinou。2013。SETSUD ¯O：面向可扩展分布式系统的基于扰动的测试框架。见 第一届*ACM SIGOPS*操作系统及时成果会议论文集，*TRIOS@SOSP 2013*，美国宾夕法尼亚州法明顿，*11*月 *3, 2013* 7 1 7 14 10 1145 2524211 2524217 。ACM，：–：。doi:。/。[33] 2014 James Lewis 和 Martin Fowler。。微服务：这一新架构术语的定义。于 2025 年 12 月检索自 https://martinfowler.com/ articles/microservices.html[34] Haopeng Liu、Guangpu Li、Jeffrey F. Lukman、Jiaxin Li、Shan Lu、Haryadi S. Gunawi和Chen Tian。2017。DCatch：自动检测云系统中的分布式并发缺陷。见 第22届编程语言与操作系统体系结构支持国际会议论文集，*ASPLOS 2017*，中国西安，*2017*年*4*月*8-12*日，Yunji Chen、Olivier Temam和John Carter（编）。ACM， 677 691 10 1145 3037697 3037735 –。doi:。/。[35] Jiangchao Liu、Jierui Liu、Peng Di、Alex X. Liu和Zexin Zhong。2022。面向微服务的在线流量录制与回放及自动模拟点识别。见 第*44*届*IEEE/ACM*国际软件工程大会：软件工程实践，*ICSE (SEIP) 2022*，*24*，*2022 22*221 230 10 1109 55303 2022美国宾夕法尼亚州匹兹堡，*5*月 -。IEEE，–。doi:。/ICSE-SEIP 。。 9793867[36] Zhenyue Long、Guoquan Wu、Xiaojiang Chen、Wei Chen和Jun Wei。2020。WebRR：用于Web应用测试的自重放增强型稳健录制/回放。见 *ESEC/FSE '20*：第*28*届*ACM*欧洲软件工程联合会议暨软件工程基础研讨会，美国线上活动，*2020*年*11*月*8-13*日，Prem Devanbu、Myra B. Cohen和Thomas Zimmermann（编）。ACM，1498–1508。doi:10.1145/3368089.3417069[37] Zhenyue Long、Guoquan Wu、Xiaojiang Chen、Chengxu Cui、Wei Chen和Jun Wei。2020。基于微服务的应用的适应度引导的韧性测试。见 *2020*年*IEEE Web*服务国际会议，*ICWS 2020*，中国北京，*10*月 *19 23, 2020* 151 158 10 1109 49710 2020 00027 -。IEEE，–。doi:。/ICWS 。。[38] Jie Lu、Chen Liu、Lian Li、Xiaobing Feng、Feng Tan、Jun Yang和Liang You。2019。CrashTuner：通过元信息分析检测云系统中的崩溃恢复缺陷。见 第*27*届*ACM*操作系统原理研讨会论文集，*SOSP 2019*，加拿大安大略省亨茨维尔，*2019*年*10*月*27-30*日，Tim Brecht和Carey Williamson（编）。ACM， 114 130 10 1145 3341301 3359645 –。doi:。/。[39] 2023 Sehrish Malik、Moeen Ali Naqvi和Leon Moonen。。CHESS：基于混沌工程的自适应系统评估框架。见 第*18*届*IEEE/ACM*自适应与自管理系统软件工程研讨会，*SEAMS 2023*，澳大利亚墨尔本，*2023*年*5*月*15-16*日。IEEE，195–201。doi:10.1109/SEAMS59076.2023.00033[40] Christopher S. Meiklejohn、Andrea Estrada、Yiwen Song、Heather Miller和Rohan Padhye。2021。服务级故障注入测试。见 *SoCC '21*：*ACM*云计算研讨会，美国华盛顿州西雅图，*2021*年*11*月*1-4*日，Carlo Curino、Georgia Koutrika和Ravi Netravali（编）。ACM，388–402。doi:10.1145/3472883.3487005[41] James W. Mickens、Jeremy Elson和Jon Howell。2010。Mugshot：面向JavaScript应用的确定性捕获与回放。见 第*7*届*USENIX*网络系统设计与实现研讨会论文集，*NSDI 2010*，*2010*年*4*月*28-30*日，美国加利福尼亚州圣何塞。USENIX协会，159–174。http://www.usenix. org/events/nsdi10/tech/full_papers/mickens-mugshot.pdf[42] Sam Newman。2015。构建微服务——设计细粒度系统，第*1*版。O'Reilly。https://www.worldcat.org/oclc/904463848[43] 2025 OpenTelemetry。。高质量、无处不在且可移植的遥测，以实现有效的可观测性。于2025年12月检索自 https://opentelemetry.io/

[44] Harendra Singh、Laxman Singh 和 Shailesh Tiwari。2022。关于测试用例优先级排序技术的系统性文献综述。国际软件创新期刊10，1（ 2022 1 36 10 4018 312263 ），–。doi:。/IJSI。Davide Spadini、[45] Mauricio Finavaro Aniche、Magiel Bruntink 和 Alberto Bac- chelli。2017。要不要使用模拟对象？：关于模拟实践的实证研究。载于第*14*届国际软件仓库挖掘会议论文集，*MSR 2017*，阿根廷布宜诺斯艾利斯，*2017*年*5*月*20*日至*28*日，Jesús M. González- Barahona、Abram Hindle 和 Lin Tan（编）。IEEE计算机学会，402–412。doi:10.1109/MSR.2017.61[46] Davide Spadini、Maurício Finavaro Aniche、Magiel Bruntink 和 Alberto Bac- chelli。2019。用于测试 Java系统的模拟对象——开发人员为何及如何使用它们，以及它们如何演变。实证软件工程24，3（2019），1461–1498。doi:10.1007/S10664-018-9663-0[47] Haley Tucker、Lorin Hochstein、Nora Jones、Ali Basiri 和 Casey Rosenthal。2018。混沌工程的商业价值。*IEEE*云计算5，3（2018），45–54。doi:10.1109/MCC.2018.032591616[48] Zhigang Wang、Lixin Gao、Yu Gu、Yubin Bao 和 Ge Yu。2018。云环境中异步迭代计算的容错框架。*IEEE*并行与分布式系统汇刊29，8（2018），1678–1692。doi:10.1109/TPDS.2018.2808519[49] Muhammad Waseem、Peng Liang、Mojtaba Shahin、Amleto Di Salle 和 Gastón Márquez。2021。微服务系统的设计、监控与测试：从业者视角。系统与软件杂志182（2021），111061。doi:10.1016/J.JSS. 2021.111061[50] Weaveworks。2016。SockShop：微服务演示应用程序。检索于2025年7月，网址：https://github.com/microservices-demo/microservices-demo [51] Thomas Welsh 和 Elhadj Benkhelifa。2021。云计算中的弹性：云领域技术综述。*ACM*计算调查53，3（2021），59:1–59:36。doi:10.1145/3388922[52] Huayao Wu、Senyao Yu、Xintao Niu、Changhai Nie、Yu Pei、Qiang He 和 Yun Yang。2023。通过容错瓶颈增强服务系统的故障注入测试。*IEEE*软件工程汇刊49，8（2023），4097–4114。doi:10. 1109/TSE.2023.3285357[53] Qinghua Xu、Guancheng Wang、Lionel C. Briand 和 Kui Liu。2025。从幻觉走向共识：利用多智能体大语言模型生成端到端测试，并提供准确的测试预言。*CoRR* abs/2506.02943（2025）。arXiv:2506.02943 doi:10.48550/ARXIV.2506. 02943[54] Tianyi Yang、Cheryl Lee、Jiacheng Shen、Yuxin Su、Cong Feng、Yongqiang Yang 和 Michael R. Lyu。2024。MicroRes：通过降级传播索引实现微服务的通用弹性剖析。载于第*33*届*ACM SIGSOFT*国际软件测试与分析研讨会论文集，*ISSTA 2024*，奥地利维也纳，*2024*年*9*月*16*日至*20*日，Maria Christakis 和 Michael Pradel（编）。ACM，325–337。doi:10.1145/3650212.3652131[55] Ding Yuan、Yu Luo、Xin Zhuang、Guilherme Renna Rodrigues、Xu Zhao、Yongle Zhang、Pranay Jain 和 Michael Stumm。2015。简单测试即可防止大多数严重故障：分布式数据密集型系统生产环境故障分析。*login Usenix*杂志40，1（2015）。https://www.usenix.org/publications/login/feb15/yuan [56] Long Zhang、Brice Morin、Philipp Haller、Benoit Baudry 和 Martin Monper- rus。2021。用于实时分析和证伪 JVM 异常处理的混沌工程系统。*IEEE*软件工程汇刊47，11（2021），2534–2548。doi:10.1109/TSE.2019.2954871[57] Man Zhang、Andrea Arcuri、Yonggang Li、Yang Liu 和 Kaiming Xue。2023。使用 EvoMaster 对基于 RPC 的 API 进行白盒模糊测试：一项工业案例研究。*ACM*软件工程与方法论汇刊 32, 5 2023 122 1 122 38 10 1145 3585009 ( )，：–：。[58] doi:。/ Xiang Zhou、Xin Peng、Tao Xie、Jun Sun、Chao Ji、Wenhai Li 和 Dan Ding。2021。微服务系统的故障分析与调试：行业调查、基准系统和实证研究。*IEEE*软件工程汇刊47，2（2021），243–260。doi:10.1109/TSE.2018.2887384[59] Hengcheng Zhu、Lili Wei、Ming Wen、Yepang Liu、Shing-Chi Cheung、Qin Sheng 和 Cui Zhou。2020。MockSniffer：刻画并推荐单元测试中的模拟决策。载于第*35*届*IEEE/ACM*自动化软件工程国际会议，*ASE 2020*，澳大利亚墨尔本，*2020*年*9*月*21*日至*25*日。IEEE，436–447。doi:10.1145/3324884.3416539