引用格式：张旭鸿，梁红，夏亦凡，等. 并行化模糊测试研究综述[J]. 信息对抗技术，2022，1(1)：24-42. [ZHANG Xuhong，LIANG Hong，XIA Yifan，et al. Parallel fuzzing：a survey[J]. Information Countermeasure Technology，2022，1(1)：24-42. (in Chinese)]

并行化模糊测试研究综述

张旭鸿[1]，梁　红[2]，夏亦凡[3]，蒲誉文[2]，纪守领[2*]

(1. 浙江大学软件学院，浙江宁波 315048；2. 浙江大学计算机科学与技术学院，浙江杭州 310007；
3. 华中科技大学网络空间安全学院，湖北武汉 430040)

摘　要　随着新一代网络信息技术的不断创新突破，软件从单机场景逐步扩展到移动终端、物联网设备、工业控制设备、云计算平台等新兴领域，推动了信息化基础设施建设的发展。然而，应用软件质量良莠不齐，给黑客组织提供了可乘之机。事件型漏洞和高危零日漏洞数量上升，如何高效准确地挖掘软件漏洞亟待解决。为实现漏洞的快速检测，模糊测试技术备受关注，它具有部署简单、自动化程度高、兼容性好等特点，能通过提供大量的输入样例实现对目标程序的脆弱性分析。现有的模糊测试通常在单处理器环境中执行，存在单个检测任务耗时长、计算资源利用率低、可持续能力差等缺陷。因此，并行化模糊测试一经提出便备受青睐。针对并行架构下的任务划分、数据存储、通信交互等问题，学术界和工业界对其展开了深入分析，并设计了一系列的实现方法。为此，系统地总结了当前模糊测试面临的挑战，概述了当前阶段模糊测试的并行化需求，着重比较分析了现存并行化模糊测试方案的优势和不足，并对高性能计算场景下并行化模糊测试的未来趋势进行了展望。

关键词　软件测试；模糊测试；漏洞检测；并行计算

中图分类号　TP 311　　**文献标志码**　A　　**文章编号**　2097-163X(2022)01-0024-19

DOI　10.12399/j.issn.2097-163x.2022.01.003

Parallel fuzzing：a survey

ZHANG Xuhong[1]，LIANG Hong[2]，XIA Yifan[3]，PU Yuwen[2]，JI Shouling[2*]

(1. School of Software Technology，Zhejiang University，Ningbo 315048，China；
2. College of Computer Science and Technology，Zhejiang University，Hangzhou 310007，China；
3. School of Cyber Science and Engineering，Huazhong University of Science and Technology，Wuhan 430040，China)

Abstract　With continuous innovation and breakthroughs in the new generation of network information technology，the software system has gradually extended from stand-alone scenarios to mobile terminals，Internet of Things devices，industrial control equipment，cloud computing platforms，and other emerging areas，promoting the development of information technology infrastructure construction. However，the software applications are of varying quality，making them vulnerable to attacks from hacker organizations. It is highly demanded

收稿日期：2022-03-08　　**修回日期**：2022-03-23

通信作者：纪守领，E-mail：sji@zju.edu.cn

作者简介：张旭鸿(1988—)，男，博士，研究员，研究方向为人工智能与安全、数据驱动软件与系统安全、大数据系统与分析；梁红(1999—)，女，博士研究生，研究方向为软件与系统安全；夏亦凡(2000—)，男，本科生，研究方向为系统与软件安全；蒲誉文(1993—)，男，博士，现浙江大学博士后，研究方向为物联网安全与隐私保护、人工智能安全、数据驱动安全；纪守领(1986—)，男，博士，研究员，博士研究生导师，国家“海外高层次人才引进计划(青年项目)”，浙江省“海外高层次人才引进计划”，研究方向为人工智能安全、数据驱动安全、软件与系统安全、大数据分析

to mine software vulnerabilities efficiently and accurately due to the increasing number of event-based vulnerabilities and high-risk zero-day vulnerabilities. To detect vulnerabilities quickly, fuzzing has attracted much attention. It finds bugs by repeatedly injecting mutated inputs to a target program with the benefit of simple deployment, high automation and compatibility. However, existing fuzzing tests are usually performed in a single-processor environment, which suffers from significant time overhead, low computational resource utilization, and poor sustainability. Therefore, parallel fuzzing has been proposed and gained much attention. Academia and industry have launched an in-depth research on parallel fuzzing and designed a series of methods for task division, data storage, and communication interaction under the parallel architecture. This work systematically summarized current challenges in fuzzing process, scientifically outlined the needs of parallel fuzzing, then focused on comparing and analyzing the advantages and disadvantages of each parallel fuzzing scheme. In the end, this work prospected for the future trend of parallel fuzzing in high-performance computing scenarios.

Keywords　software testing; fuzzing; vulnerability detection; parallel computing

0　引言

随着新一代信息技术的飞速发展，软件广泛应用于个人生活、企业管理、工业生产和政府服务等各个方面，深刻影响着生产和生活方式，推动了信息化基础设施建设的发展。然而，大多数厂商开发软件时缺乏完善的程序测试和补丁修复管理机制，导致软件漏洞的快速检测和识别困难重重[1]。一旦软件漏洞被不法分子恶意利用，就可能造成巨大的政治和经济损失[2]。目前大部分严重的网络安全威胁都是由安全漏洞引起的，仅 2021 年上半年，国家信息安全漏洞共享平台收录高危漏洞的数量就达 3 719 个，涉及政府机构、重要信息系统等网络安全漏洞事件近 1.8 万起[3-4]。美国网络战司令部在其 2020 年年度技术挑战报告中，也将软件漏洞列为网络空间防御的第一大技术挑战[5]。因此，提高软件漏洞挖掘能力、完善软件产品服务测试是推动软件产业链升级、提升产业基础保障水平、形成软件协同产业生态的重要保障，在国防建设和国民经济中占据了重要地位。

为了实现软件缺陷的早发现、早防御、早弥补，构筑安全闭环、链路可靠、安全稳固的数字化软件生态环境，学术界和工业界提出了多种软件分析方法。其中，模糊测试技术因具有部署简单、自动化程度高、兼容性好等特点而备受青睐。根据测试样例的生成方式，模糊测试可以分成基于规范模板生成的模糊测试[6-9]和基于突变算法的模糊测试[10-12]两大类型；以工具对目标程序的内部结构知悉程度为划分标准，模糊测试通常被细分为黑盒模糊测试[13-16]、白盒模糊测试[17-20]和灰盒模糊测试[21-23]3 种类型。随着软件应用场景的不断丰富，模糊测试技术呈现出纵深多元、智能优化、多领域渗透的发展趋势，学者们也在模糊测试的系统性能优化、评估体系构建和应用场景拓宽等方面展开了深入研究。

现有的模糊测试通常在单处理器环境上启动单个工具实例，持续运行长达数天甚至数月，存在检测任务耗时长、计算资源利用率低、可持续能力差等缺陷。面对现代复杂的大规模软件系统，依靠单核机器开展的模糊测试在性能和鲁棒性指标上存在明显不足[24-25]。随着多核架构的兴起，并行化计算成为提高运行速度、降低时间成本的常见渠道，并行化模糊测试思想也备受关注。学术界和工业界对其展开深入分析，设计实现了多种并行化模糊测试方案[24-42]。因此，亟需对现有工作进行科学系统地归纳梳理，为后续开展并行化模糊测试研究提供指导意见。

本文首先详细阐述了模糊测试的基本逻辑和工作流程，总结了当前模糊测试在性能效率提升、评估体系构建、多元场景涵盖等方面面临的挑战，重点阐明了并行化模糊测试场景的研究动态，归纳了并行化模糊测试的需求背景，从单机单原型、单机多原型、多机单原型、多机多原型 4 个方面进行分类，整理了各并行化模糊测试方案的核心思路和设计架构，讨论了现有并行化方案

的先进性和局限性，最后展望了高性能计算场景下并行化模糊测试未来的发展方向。

1 模糊测试基本流程和存在的问题

模糊测试技术由来已久，早在 20 世纪 90 年代初就被用于测试 Unix 程序的健壮性[43]。理论上，模糊测试根据启发式规则或者随机变异策略生成大量输入样例，通过监控目标程序的执行情况来报告引起程序崩溃的异常行为，实现软件漏洞的自动化检测。模糊测试在漏洞利用生成和渗透测试等场景中率先得到了部署使用[44-45]。在由美国国防部高级研究计划局举办的网络大挑战中，诸多团队使用模糊测试技术取得了很好的成绩[46-48]。随后，软件测试领域掀起了模糊测试的研究热潮，Adobe[49]、Cisco[50]、Google[51-53]和 Microsoft[54-55]等知名供应商开始采用模糊测试衡量产品的安全性。

模糊测试的整体流程大致相似，普遍以初始测试样例集和目标程序作为输入，以检测到的触发目标程序异常崩溃或超时错误的测试样例集作为输出，通过预处理、测试样例生成、目标程序执行与状态监控、能量调度分配、测试样例质量评估等环节循环往复来实现自动化的动态漏洞检测。模糊测试工具典例 American Fuzzy Lop[56]（AFL）的基本工作流程如图 1 所示。

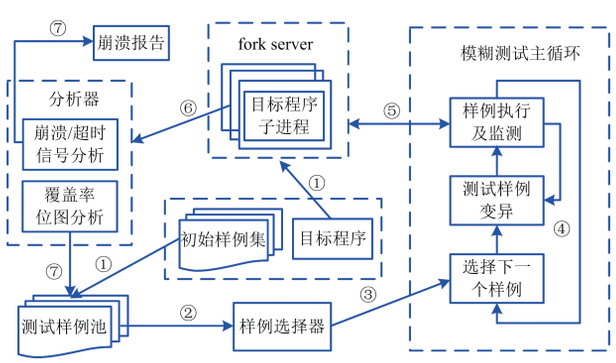

图 1 AFL 工作流程

Fig. 1 Illustration of AFL workflow

AFL 工作流程由以下 7 个步骤构成。

（1）AFL 对目标程序进行插桩，利用位图和共享内存机制实现边覆盖率信息的获取，为代码覆盖率和程序状态的反馈做准备。与此同时，AFL 会读取初始样例集里的所有文件构造测试样例池，采用队列结构存储以便快速执行对测试样例的增加、替换、变异等操作。需要说明的是，上述工作只需在开启模糊测试任务时运行 1 次即可；

（2）经过初始化操作后，AFL 根据当前队列中测试样例（又称种子）的执行时间和文件大小，为每条路径标注最优测试样例；

（3）AFL 通过能量分配为种子设定变异次数；

（4）AFL 根据变异次数随机选择变异操作，对种子实施变异从而生成新的测试样例；

（5）当生成新测试样例后，AFL 以该样例为输入，执行插桩后的目标程序并监控程序状态；

（6）在执行完成后，采用 fork server 机制收集到的覆盖率和超时崩溃信号将被传递给分析器；

（7）如果程序发生崩溃或超时，则将引发崩溃或超时的样例存入本地并进行崩溃报告；如果测试样例触发了新的边覆盖率信息，那么该样例将加入到测试样例池中，连接在队列尾部。

AFL 后续将循环执行步骤（2）～步骤（7），直到使用者中断模糊测试任务为止。

随着信息化的发展，软件规模不断增大，软件结构日趋复杂，应用场景也纷繁多样，对模糊测试技术提出了更高的要求。学术界和工业界研究人员通过系统性研究，提出了一系列解决方案，但当前研究更多集中于单核场景，工业落地的实际效率存在明显的局限性，具体分析如下。

（1）性能效率提升。模糊测试技术虽然通过动态执行来保证了缺陷检测的低误报率，但覆盖率低、耗时长一直是限制模糊测试在工业界广泛落地的瓶颈，在有限时间内穷尽所有可能的执行状态很难实现。为了提升模糊测试的性能和效率，研究人员提出了多种优化思路，也取得了显著的提升效果。从模糊测试的整体流程出发，一方面，提高优质种子的利用率是提高可用性的常见解决思路[10-11,57-60]；另一方面，目标程序执行速度直接影响模糊测试的效果，硬件加速、自动化选择最佳编译方案、过滤无效的代码覆盖信息追踪都可显著提高系统效率[61-64]。从多种软件分析技术融合的角度来看，符号化执行[46,65-69]、污点分析[18,70-73]等通用程序分析技术辅助模糊测试取得了一定的成效，GreyOne[71]利用字节级变异推测输入和分支变量的依赖关系，采用一致性引导的进化策略来筛选优质种子；RedQueen[73]利用输入到状态同步的性质来解决魔术字节以及校验和的匹配问题。从人工智能赋能的角度来看，

数据驱动为模糊测试优化开辟了新思路[10,72,74-78]。Fuzzguard 通过深度学习来预测程序输入的可达性，解决了标签数据不平衡和训练过程中时间不足的问题[74]；Neuzz 使用前馈神经网络有效地建模目标程序的分支行为，随着训练数据迭代改进代理模型[76]。不可否认，上述方案对于模糊测试性能和效率提升都起到了一定的作用，但是普遍集中在单核场景下，计算资源的利用率和分析的可扩展性受到限制，存在改进的空间。

（2）评估体系构建。模糊测试工具的多样性和目标程序的复杂性也给规模化测试带来了新的挑战。一方面，目前模糊测试工具种类丰富，相互之间的边界并不清晰，现有性能分析缺乏统一的标准；另一方面，由于目标程序的多样性，不同测试工具在不同目标程序、甚至同一程序的不同模块测试上的表现也可能不完全相同。在模糊测试评估体系构建方面，相关研究不断完善，为实际测试中模糊测试工具的选择提供了一定的理论依据[79-84]。Li 等[79]总结了独特漏洞数量、独特漏洞质量、漏洞发现速度、漏洞发现稳定性、覆盖率、开销 6 个评估指标对模糊测试工具进行了综合评估；Wang 等[83]侧重谈论了不同代码覆盖率统计方式对模糊测试的影响，在理论上给出了比较不同代码覆盖率的指标；Klees 等[84]从基准算法、测试目标程序、评估标准、初始测试集选择、重复次数、崩溃去重等多个角度给出建议，说明了如何保证模糊测试的实验结果可靠有效。虽然模糊测试工具的测试集构建和性能评估上已经有相对成熟的研究，但普遍忽略了模糊测试工具之间的相似性、协同性分析，在大规模并行测试场景下的模糊测试工具选择问题仍有待解决。因此，构建完善的评估体系来指导实际测试中工具选择成为了模糊测试实用化的重要前提。

（3）多元场景涵盖。区块链、人工智能、物联网等新技术促进了软件形态的更新换代，复杂系统缺陷分析、多点触发漏洞检测等问题给模糊测试提出了新的要求。由于用户接口黑盒、部署环境差异、探索空间激增等特性，面向传统软件的模糊测试技术难以直接应用，多元场景下的可用性和高效性是模糊测试技术发展的又一挑战。研究人员逐渐突破传统软件测试的思维桎梏，推动了模糊测试技术在软件供应链[85-86]、操作系统内核及文件系统[67,87-93]、虚拟机管理程序和外设及 CPU 芯片[94-97]、物联网设备及协议设计[98-101]、区块链智能合约[102-105]、数据库管理系统[106-107]、JavaScirpt 及 DOM（document object model）引擎[108-111]、编译器及 SMT（satisfiability modulo theories）求解器[9,112-114]、机器学习模型或系统[115-117]、云服务 API（application programming interface）[118-119]等场景下的蓬勃发展。在虚拟机管理程序的测试上，V-SHUTTLE[94]通过解耦化嵌套结构并启用类型感知设计了语义感知的测试框架，实现了深层代码的自动化探索。在浏览器引擎测试上，Freedom[109]和 DIE[110]则考虑了特定的语法语义要求，利用抽象语法树、中间语言等技术辅助样例生成来提高输入样例的有效性。虽然研究人员巧妙利用了不同场景的特征进行适配，提出了一系列的解决方案，但是新兴场景下模糊测试的时间成本明显增加，如何利用并行技术来推进大规模测试从而确保在用户可接受的时间预算内返回结果尚待探究。

2　并行化模糊测试

作为软件测试分析领域的核心技术之一，模糊测试技术被不断夯实完善，在诸多领域取得了长足的进步，但仍不足以支持对现实中大规模系统的快速充分测试。基于传统模糊测试无法满足用户对海量软件的安全测试需求这一现实背景，并行化模糊测试技术作为一个颇具前景的研究方向被提出，以解决传统模糊测试性能不足的问题。

2.1　并行化模糊测试需求背景

并行计算是提高计算机系统计算速度和处理能力的有效手段，核心目的在于优化性能，主要思想在于将复杂问题拆分成若干部分，各部分均由独立的处理器或计算资源进行计算，从而提高效率[120]。并行化模糊测试的重要性主要体现在以下方面：

（1）软件规模的急剧增加与迭代更新的日益频繁造成了高工作量的问题。随着信息化与现代化建设的推进，软件功能和结构日趋复杂，百万行、千万行甚至更大规模的软件系统已不少见，对单个软件实现功能全覆盖的模糊测试时间开销随之增大。同时，软件迭代更新速度的加快不仅使得测试任务增多，而且对单次测试任务的

时效性要求更高。在实际应用中，用户往往对测试结果有较高的时效性要求，希望能在尽可能短的测试时间内获取较多的反馈信息。例如，在新版本发布前，用户可能希望完成一轮相对完整的测试，以发现可能新引入的漏洞。

(2) 新兴场景下复杂漏洞的检测时间成本明显增加。以 Linux 内核的并发错误测试为例，涉及的系统调用数量超过 400 个，每个系统调用参数数量和设置种类繁多，作为模糊测试输入的系统调用序列长度没有上限，再加上线程交错情况复杂，因此模糊测试实验时间普遍持续长达数周。相似地，针对 JavaScript 引擎中 JIT(just-in-time compiler)加速优化引起的漏洞测试通常采用分布式架构测试来减少时长，即便如此，对流行浏览器的充分测试往往长达数天。

(3) 提升性能的研究不计其数，如何集成多种工具实现高效测试亟待解决。不同模糊测试工具的优化思路不同，表现效果存在差异。Li 等[79]证明了没有一个模糊测试工具能够在所有方面优于其他工具。具体来看，现有模糊测试工具在设计时大多只针对某类测试目标，在特定条件下性能最佳，比如人工注入漏洞的 LAVA-M 测试集[121]所包含的可执行程序就存在编程逻辑较为简单、大部分注入漏洞与魔术字节具有强相关性的特点，兼备符号执行、污点分析等通用程序分析技术的模糊测试工具[65-73,122]在该测试集上表现普遍优于其他工具。

(4) 多核架构下性能提升显著，如何提高模糊测试任务的计算资源利用率有待探索。单核 CPU 硬件性能若想达到显著提升，同代制造工艺限制下只能提高频率，但仅提高主频是不可持续的，主频越高意味着功耗越高、发热也越严重，引入的设计、制造、验证成本开销巨大。硬件厂商纷纷转向多核产品的研发，采用多核设计性能显著优于高主频的单核产品，在发热和功耗上更不可同日而语。未来低功耗、高性能的多核 CPU 必将更加流行，要想充分利用计算资源，就必须扭转单核模糊测试占据主流的局面，面临并行化转型发展的迫切需求。因此，并行化模糊测试的目标是充分利用高性能芯片等性能显著提升的计算资源，通过多实例高效协同的并行方案进一步提高单位时间内的综合测试效率，有效减少单个模糊测试任务的时间开销。

2.2 并行化模糊测试研究现状

随着用户对模糊测试工具性能要求的不断提高，原有单任务模式下的模糊测试无法满足即时获取测试结果的需求，并行化模糊测试应运而生。早在 2010 年，Xie[123]就提出了基于网格计算的大规模并行模糊测试想法，利用网络中多台计算机的资源来支持应用程序的模糊测试，实现了分发、测试以及数据收集等流程；随后，Wang[124]借助二进制代码的静态分析方法，实现了基于异步编程模型的分布式文件模糊测试系统；Lee 等[125]设计了模糊测试引擎评估工具 FEET，构建了移动平台安全测试库 STAMP，形成了移动设备黑盒测试场景下的分布式并行模糊测试框架 MVDP；Beterke[126]沿用了中心节点向工作节点分发任务的思想，实现了覆盖率引导的可扩展分布式模糊测试工具 EvoFuzz。

同一时期，很多研究人员关注到了白盒模糊测试与混合模糊测试的并行利用问题[127-130]，分别开发了基于计算节点簇的分布式测试系统 Cloud9[128]、面向文件解析类程序的并行化模糊测试工具 SAGE[129]、基于混合符号执行路径取反算法和复合化用例生成方法的并行化智能模糊测试系统“谛听”[130]。类似地，开源框架 Sulley[7]、商业工具 Peach[16]、Defensics[131]等模糊测试工具在开发之初就考虑到了并行化场景的设计，支持特定任务的大规模测试。总体上，2017 年之前的并行模糊测试研究相对分散粗浅，并未形成完整的体系，存在很多的改进空间。为了完善和利用并行化模糊测试技术，研究者在早期探索的基础上继续深入研究，取得了可观的进展。

本文调研了近 5 年的相关研究，从应用场景、架构设计、指导信息同步粒度、任务划分粒度、开源情况、测试数据集和其他特征 7 个维度归纳整理结果见表 1 所列。尽管现有工作研究目的各不相同，实现细节多样，但总体来看，几乎所有的工作都明确了其应用场景以及架构设计。依据并行模糊测试场景中是否存在分布式网络通信，以及架构设计中是否使用了多种模糊测试工具，本文将现有的并行模糊测试研究划分为 4 个类型：单机场景下的单原型架构、单机场景下的多原型架构、多机场景下的单原型架构以及多机场景下的多原型架构。

2.2.1 单机场景下的单原型架构

单机并行模糊测试主要考虑单台多核机器工

作场景，在该场景下，用户根据本机资源，在本地启动多个模糊测试任务实例，并在实例之间制定一系列协作机制。单机并行模糊测试的所有工作阶段都在本地完成，具有额外开销小、实现相对简单等特点。在并行模糊测试研究早期，几乎所有相关研究都围绕单机单原型并行模糊测试开展。

表 1　近 5 年并行模糊测试研究分类

Tab. 1　Classification of research on parallel fuzzing in the past five years

方法名称	应用场景		架构设计		信息同步粒度	任务划分粒度	开源情况	测试数据集	其他特征
	单机	多机	单原型	多原型					
Zou Y, et al.[24]	√	×	√	×	粗	中	×	G/L/R	
ENFUZZ[25]	√	×	×	√	粗	未涉及	×	G/L/R	
Xu W, et al.[26]	√	×	√	×	中	未涉及	×	G	
Lian M, et al.[27]	√	√	√	×	未涉及	未涉及	×	—	
PAFL[28]	√	×	√	×	细	中	×	G	
Li Y, et al.[29]	√	√	√	×	中	未涉及	×	—	静态分析技术
Ye J, et al.[30]	√	×	√	×	中	中	×	R	
P-Fuzz[31]	×	√	√	×	中	中	×	L/R	
Diskaller[32]	√	√	√	×	中	未涉及	×	Linux v4.16	内核模糊测试
Cupid[33]	√	×	×	√	未涉及	未涉及	√	L/G	研究原型选取
Roving[34]	×	√	√	×	粗	未涉及	√	—	工具原型
Li Y, et al.[35]	√	√	√	×	中	未涉及	×	R	静态分析技术
OSS-Fuzz[36]	√	√	√	√	粗	粗	√	—	工业界
OneFuzz[37]	√	√	√	√	粗	粗	√	—	工业界
AFL-EDGE[38]	√	×	√	×	粗	中	×	R	静态分析技术
AFLTeam[39]	√	×	√	×	未涉及	细	√	R	静态分析技术
Tang R, et al.[40]	√	×	√	×	中	中	×	L/R	
UniFuzz[41]	×	√	√	×	中	中	×	R	
CollabFuzz[42]	√	√	×	√	中	未涉及	√	G	
AFL-P[56]	√	√	√	×	粗	未涉及	√	—	

注：① 指导信息同步粒度分级，同步种子文件为粗粒度，同步运行覆盖率为中粒度，同步额外定制信息为细粒度；② 任务划分粒度分级，根据软件模块划分为粗粒度，通过种子分发或变异策略划分为中粒度，根据程序结构与执行状态划分为细粒度；③ “未涉及”指研究中未明确给出具体算法；④ 测试数据集，L：LAVA-M 数据集，G：Google's fuzzer-test-suite，R：随机选取的程序集合，“—”：未提及；⑤ 应用场景，专门应用场景在“其他特征”中注明，若未注明则默认为通用型模糊测试工具；⑥ AFL-P 即 AFL 并行模式。

AFL 是已知最早提出的系统性模糊测试工具，也是现今部署最广泛的模糊测试项目。自发布以来，AFL 已经从真实世界的复杂软件中发现了大量重要的安全漏洞。为了更好地发挥多核设备的性能，提升模糊测试的速度，AFL 预置了默认的并行模式，以实现多个任务实例的协同工作，并行模式架构如图 2 所示。

在该模式中，系统将启动多个 AFL 实例，每个实例在公共根目录下各自维护私有的本地目录，用来保存测试任务过程中产生的测试用例。在理想状态下，设计者希望并行模式下的 AFL 性能可以随着可用资源的增加呈线性增长。然而，如果各实例完全独立运行而不进行交互，那么实际使用中对性能的提升非常有限。为此，AFL 明确了指导信息同步这一关键步骤，在该步骤下，各实例将从其他实例中获取有价值的测试用例文件，从而更好地协作完成测试任务。值得一提的是，指导信息同步机制的设计在后续并行模糊测试研究工作中被一直延续，且同步粒度不断细化，诸如代码覆盖率，崩溃信息等。

AFL 并行模式同步机制算法如图 3 所示，各 AFL 实例在完成常规测试循环，即“生成测试用例、启动单次执行、收集反馈信息”这一过程结束后，将逐一扫描其他实例的本地目录，该目录下保存了每个 AFL 实例在工作流程中产生的有效测试用例。为了避免每次扫描到相同的内容，

AFL 以一个严格排序的序号为每个测试用例命名，并记录上一次扫描中最后访问到的测试用例序号，作为下次扫描的起点，这一阶段称为同步阶段。在扫描过程中，各实例将依次执行其他实例目录下新生成的测试用例，以获取其执行反馈。如果在执行一个测试用例时程序覆盖到了本实例此前未发现的新逻辑分支，那么该实例会把这个覆盖了新分支的测试用例保存到自己的本地目录下，以待进一步利用。AFL 并行模式的同步设计使不同的实例之间可以共享有价值的测试用例，从而节约了每个实例在庞大的输入空间中冗余的探索。然而，算法中每个实例需要循环遍历其他实例文件目录，这一步骤的计算复杂度高达 $O(n^2)$，严重影响了并行化性能。

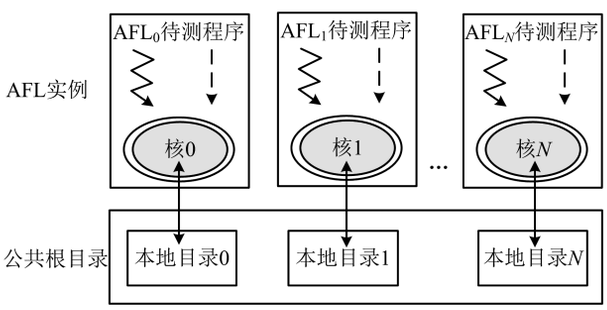

图 2 AFL 并行模式架构

Fig. 2 Architecture of AFL parallel mode

作为并行模糊测试的先行者，AFL 通过在文件系统中遍历其余实例的种子目录，将独特的测试用例同步到本地，并重复执行新添加的种子来获取其中的指导信息。这种简单直观的做法尽管起到了一定的效果，但文件检索和重复执行过程引入了过多的开销。为了优化 AFL 原始并行模式，尽量控制同步开销，Xu 等[26]为原始 AFL 并行模式设计了新的操作原语，提出了基于测试日志的队列结构。测试日志是一种处于内存中的数据结构，每个实例的测试日志中保存了该实例历史执行的测试用例文件名以及覆盖率位图。借助内存中的测试日志，各实例可以通过入队操作存储本地历史执行信息，通过出队操作读取其他实例的历史执行信息。增加新的操作原语后，一方面，由于读取内存的效率远高于文件 I/O 的效率，将同步信息保存在内存中而非存入到硬盘文件上，可以有效降低硬盘文件读取引入的开销；另一方面，直接将覆盖率信息以位图格式保存在内存中，减少了重复执行测试样例引入运行开销。实验结果表明，新的操作原语显著提高了并行模式下系统执行速度。

Algorithm 1: AFL parallel mode

```
Input:Fuzzer instances list FuzzerInstances[]
repeat
  for each instance i in FuzzerInstances[] do
    TypicalFuzzingProcess();
    // run normal fuzzing process
    // after that AFL will do seed synchronization
    Neighbors = i.neighbors();
    // Neighbors contain all other instances
      foreach neighbor n in Neighbors do
        LastIndex = i.checkLastIndex(n);
        // Scan new seeds generated after we last seen
        SeedPool = n.seedpool;
        foreach seed s in SeedPool do
          if s.index > LastIndex then
            Cover = i.run(s);
            if i.newCover(Cover) then
              i.seedpool.push(s);
        end
      end
  end
until timeout or abort-signal
Output : Crashing seeds crashed
```

图 3 AFL 并行算法设计

Fig. 3 Algorithm of AFL parallel mode

Xu 等人有效地降低了 AFL 并行模式的开销，但其设计思路局限在原始 AFL 工作流程中，普适性相对较差。近年来的前沿模糊测试工具往往增加了定制化的额外指导信息，用于优化其工作流程。例如，AFLFast[21]需要收集路径触发频率等信息来调整种子的变异次数，FairFuzz[132]记录了每次测试运行中最少被触发的代码分支。传统的同步机制设计，如 AFL 并行模式中仅同步种子文件，不考虑路径触发频率、代码分支触发次数等信息的收集更新，核心优势难以充分发挥。同时，由于这些优化版本在开发过程中直接沿用了 AFL 的并行模式设计而未作任何修改，导致测试并行执行时平均性能大幅度下降。为了实现额外指导信息的同步，PAFL[28]设计了一种称为全局指导信息的数据结构，在实例之间共享。在这一设计下，各实例除了需要维护本地种

子库及指导信息之外,还共享全局种子库与全局指导信息。当实例完成阶段性的测试后,会将本地的指导信息上传至全局,随后获取其他实例上传的指导信息作为本地信息的补充。此外,PAFL 较早明确了任务冲突这一普遍存在于并行模糊测试中的挑战,即在单原型并行模糊测试过程中,由于不同的实例都基于相同的设计,且遵循经典模糊测试工作流程,这些实例可能会在同一时刻执行相同的任务。具体来说,不同的实例可能会对同一个种子进行变异,并生成相同的测试用例。这种冗余的工作会造成较大的资源浪费,严重影响并行模糊测试的性能。为了缓解任务冲突的性能损害,PAFL 设计了一种简单可用的任务划分算法,要求每个实例只关注特定的位图区域,从而避免任务冲突。对于 N 个模糊测试工具实例,PAFL 将位图根据序号划分为 N 个相邻的区域,并将该区域分配给对应的实例。在执行过程中,实例将读取目标区域执行信息来判断测试样例是否满足条件,对不满足条件的样例予以抛弃。通过全局指导信息的共享和基于位图的任务分配策略,PAFL 有效增强了 FairFuzz 和 AFLFast 并行模式下的性能,分支覆盖率和漏洞触发情况都有明显的提升。

PAFL 明确了任务划分算法的概念并提供了相应的算法实现,但简易的算法设计致使很多重要信息的丢失,其性能尚不能满足用户的期望。为进一步优化设计,AFLTeam[39] 将静态分析技术与并行模糊测试相结合。AFLTeam 指出,由于位图中的序号是完全随机分配的,一个区域内相邻的内容之间大部分并没有结构上的联系,这使得不同实例的任务过于分散;同时,在模糊测试的过程中,测试工具可以通过执行过程中的反馈获取重要的程序结构信息,因此设计者希望能够利用实时反馈动态进行任务划分。AFLTeam 对待测程序进行静态分析来构建程序调用图,在程序调用图中保存了丰富的结构信息。AFLTeam 随后使用图划分算法将程序调用图划分为多个函数级子图,每个子图对应一个任务,分配给不同实例。在测试过程中,实例将根据自己的任务筛除不包含目标函数的种子,同时对已完全覆盖的任务进行合并,实现动态更新。AFLTeam 为任务划分研究开拓了新的思路,更多静态分析技术可能被引入用于解决该问题。

除了以上详述的前沿工作,单机单原型并行模糊测试领域还有诸多值得研讨的工作。同样关注并行模糊测试中的任务冲突问题,Ye 等[30] 将一组种子的变异与执行定义为任务,依据程序状态进行任务划分,认为只要种子触发的程序状态各不相同,就可以保证任务之间各不冲突。为了识别程序状态,Ye 等选取可以触发独特程序分支的种子,并将这些特殊的种子分发给不同实例。各实例随后对接收到的种子执行约束求解,以保证变异阶段能够保留独特的程序分支。无独有偶,AFL-EDGE[38] 同样根据测试用例在执行过程中的状态信息划分不同的任务。AFL-EDGE 根据种子的边覆盖率将其所有种子划分为互斥且轻量级的子集,并交付给不同实例。每隔一个周期,AFL-EDGE 将重新收集所有种子,执行再次划分与下发。该工作指出,各分组内种子的边覆盖率各不重合,在执行过程中可以有效避免任务冲突。针对基于变异的并行模糊测试中实例间的任务冲突问题,Zou 等[24] 从变异策略角度出发,构建了以变异策略权重模型为基础的差异化并行调度模型,设计了多模糊测试实例间基于有效样本同步的协同优化方法,定期同步各测试实例间的有价值测试样例并且调整不同变异策略的应用比例,减少模糊测试实例间的测试冗余,提高系统的综合覆盖率。类似地,Tang 等[40] 也考虑到了并行场景下的种子变异优化问题,提出了一种基于两阶段变异的并行模糊测试方法。首先将随机突变运行一段时间来检测目标程序的状态空间,并把检测到的状态空间划分为独立的状态子空间,然后将子空间探索任务分配给每个模糊测试实例,并在各个实例上使用相同的有序变异。这些做法有效缓解了并行模糊测试中存在的生成测试用例冗余度高、综合覆盖率低等问题。

作为并行模糊测试的研究基石,单机单原型架构设计的百家争鸣为后续研究者提供了丰富的素材,开辟了指导信息同步、任务分配、变异策略调度等多种优化方向,也取得了显著的提升效果,使后续研究向分布式、多原型场景的扩展成为可能。

2.2.2　单机场景下的多原型架构

承前所述,多原型即在并行测试中,运行多个种类不同的模糊测试实例。多原型架构源于复杂软件系统程序测试的用户需求。具体来看,

不同模糊测试工具优化思路不同，在不同测试对象上的表现不同，如 SlowFuzz[60] 为需要更多计算资源的种子赋予较高的优先级，在算法复杂性漏洞的测试上效果更佳；Driller[46] 和 QSYM[69] 使用了具体符号执行技术，在求解复杂的路径约束问题上颇具优势。当面对具有多个模块的大型工业软件时，单一原型不再能够全面有效探索待测软件的输入空间，并行模糊测试的性能提升遭遇瓶颈，难以发挥绝对优势。为了结合不同模糊测试工具的优势，多原型架构应运而生，并在单机场景中率先起步。

ENFUZZ[25] 首次提出了多原型并行模糊测试的概念，指出原型选择应当尽可能保证多样性，从而降低不同原型实例下模糊测试子任务的耦合程度，实现系统资源的高效利用。ENFUZZ 从测试用例生成方式、种子选择和变异策略、覆盖率粒度 3 个方面人工筛选了 AFLFast[21]、FairFuzz、libFuzzer[133]、Radamsa[6]、QSYM 和 AFL 这 6 种模糊测试工具作为备选，从覆盖率和测试样例 2 个维度考虑设计了基于全局异步和本地同步的种子同步机制，实现了多个模糊测试工具之间的信息定期同步。ENFUZZ 架构设计[25] 如图 4 所示，当准备对目标程序开展模糊测试时，ENFUZZ 首先从备选模糊测试工具中选择若干作为基础模糊测试器，将模糊测试器与种子同步机制结合，监控基础模糊测试器的状态并在彼此之间共享有趣种子，最后收集崩溃和覆盖率信息并给出报告。实验表明多原型设计在路径覆盖、分支覆盖、错误发现方面都比单原型表现更佳。

直观上，选择一个好的模糊器组合是最大化协作模糊过程整体性能的重要步骤。但美中不足的是，ENFUZZ 中原型选择依赖人工的经验知识指导，限制了自身的实用性和可扩展性。针对这一现象，Cupid[33] 提出了一种用于计算多个模糊器协作程度的互补性指标，在有限的时间段内，隔离运行单个模糊工具并收集分支覆盖频率信息，利用简单的概率模型实现了模糊测试工具互补性的自动预测。Cupid 通过对不同数据集进行超过 40 000 个 CPU 小时的广泛评估，表明了该方案在分支覆盖率、错误发现和发现覆盖率的延迟方面明显比 ENFUZZ 提供的模糊测试工具选择组合少，与指数增长的穷举搜索相比，计算时间减少了多个数量级。

单机场景下的多原型架构模糊测试研究不仅涵盖了单机单原型架构下的并行性能优化研究，还引入了一个全新的研究方向。即如何利用基于不同设计的模糊测试工具的协同，发掘单个模糊测试工具无法触发的程序漏洞。尽管该方向衍生的工作目前较少，但本文认为这一类多原型架构的研究可以促进前沿研究工作有机结合，求解大型软件系统不同模块中的复杂路径约束，具有极高的实用价值。

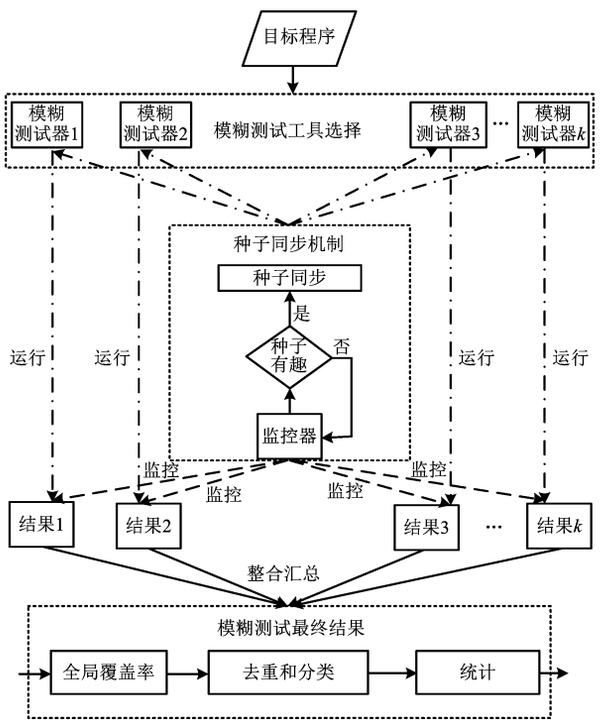

图 4 ENFUZZ 架构设计

Fig. 4 Overview of ENFUZZ

2.2.3 多机场景下的单原型架构

随着互联网高速发展，软件系统也越来越庞大，仅仅在单台机器上运行并行模糊测试逐渐无法满足开发者对大规模软件系统的测试需求。计算资源的日渐丰富推动着众多分布式系统的开发与推广，多机场景下的并行模糊测试也被提上日程。显而易见的是，直接将单机并行模糊测试方案迁移到多机场景下效果并不乐观，高额的网络通信开销使很多在单机场景下表现良好的方案不再适用。因此，研究者开始着重攻克多机场景下的并行模糊测试这一难题，绝大多数相关研究集中在同类原型实例的部署，即多机场景下的单原型架构。

Roving 是一个具有代表性的基于 AFL 的轻

量级分布式框架，由一个主控节点和若干工作节点构成，可以支持数十个客户端同时运行。工作节点负责 AFL 的运行，每隔固定时间（在 Roving 中默认为 300 s）将处理结果上传到主控节点[34]。然而，Roving 仅仅在文件级别同步了崩溃文件与输入队列，也未涉及任何任务划分策略。这些缺陷严重影响着 Roving 的效率，使其无法大规模推广使用。

P-Fuzz 是早期分布式单原型架构设计中相对成熟的典例[31]。它在 Roving 等框架的基础上提出了分布式场景下并行模糊测试的 2 个重要问题：（1）在分布式场景下，如何有效同步指导信息并避免数据竞争？（2）如何在考虑不同节点算力的情况下，实现分布式环境中的任务划分？

对于这些问题，P-Fuzz 设计了定制化的 Server-Client 工作模式，P-Fuzz 架构设计[31]如图 5 所示。测试用例执行覆盖率是并行模糊测试过程中的重要指导信息，它需要实时被各实例读取并更新。P-Fuzz 通过同步位图共享指导信息——其在中心服务器上建立了一个数据库，用于保存种子文件和覆盖率位图。各实例实时上传与下载位图，实现全局覆盖率信息的同步。然而，如果不对覆盖率的上传加以控制，可能会导致不同实例上传的位图彼此覆盖，丢失重要信息。为了消除该数据竞争，P-Fuzz 在中心节点维护了一个位图队列，当队列中存在多个位图时，中心服务器将使用“与”计算将位图信息混合，从而避免数据丢失。

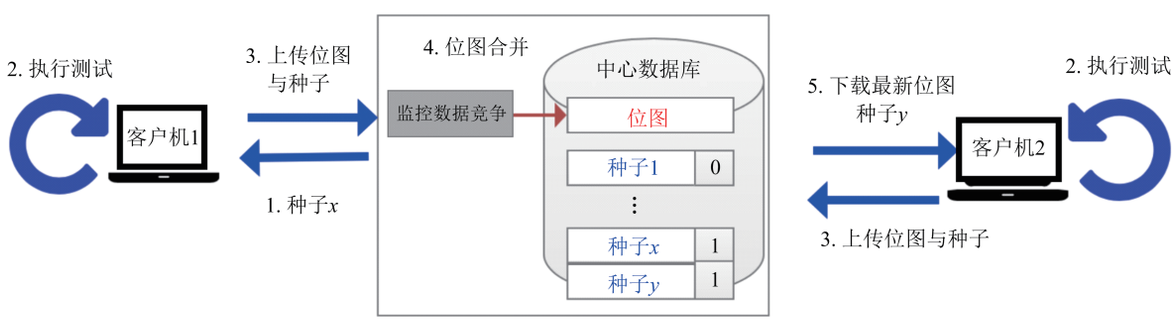

图 5　P-Fuzz 架构设计

Fig. 5　Overview of P-Fuzz

为了避免绝对的任务冲突，P-Fuzz 要求不同的客户机在同一时刻只能访问数据库中的不同种子。这一规则通过对正在使用的种子文件添加共享锁实现，从图 5 可以看出，正在被访问的种子文件标签为 1，实例只会下载标签为 0 的种子。此外，客户机被要求每次仅下载 1 个种子，这使得大型测试任务被划分为以种子为单位的微任务。由于客户机实例只会在当前微任务完成后请求下一个任务，这一概念有效缓解了分布式场景下设备算力不均衡的问题。在实现细节上，P-Fuzz 有很多可取之处。具体来说，P-Fuzz 还会在运行过程中实时监测客户机情况，实现简易的容灾处理。P-Fuzz 为不同客户机实例分配不同的变异算法进一步避免了相对的任务冲突，这可以视作多机多原型架构的早期雏形。虽然 P-Fuzz 在开销上的权衡在后续工作中被证明并不完全精确，但其思想仍可为后续研究提供宝贵建议。

UniFuzz[41]在 P-Fuzz 上做了进一步扩充，指出了 P-Fuzz 同步机制的关键缺陷，即频繁更新的位图和种子文件的上传与下载为系统带来了大量额外的负荷、微任务缺失统一的调度机制使测试性能不够稳定。UniFuzz 按照共享所需的开销，将信息从小到大分为 3 类：测试状态信息、种子文件、覆盖率信息，分别设计了对应的同步方式。对于通常只有数个字节，但需要频繁更新的状态信息，UniFuzz 在网络通信中直接传递。对于文件大小达到 kb 级的种子，UniFuzz 依靠中心数据库，各实例间只共享种子文件名，待需要使用时再从中心数据库中下载种子文件。UniFuzz 直接放弃了 P-Fuzz 中频繁的覆盖率信息共享，通过在客户机本地重复执行测试用例获取对应的覆盖率信息。

针对并行测试中大规模和多场景的需求，Lian 等[27]提出了一种基于云平台的动态资源感知并行化模糊测试框架，通过四级流水线并行结构与多层次并行度动态调整的资源配置策略，有效提高了并行模糊测试的并行处理能力和资源利用率。针对大规模并行测试中节点易发生故障的问题，Lian 等也提出基于优先级调度的容错

处理方法，实现系统的有效容错。面向内核场景下的漏洞检测问题，Diskaller 构建了以计算节点和控制节点组成的星型并行模糊测试结构[32]。各计算节点由管理器和执行器两部分构成，管理器利用远程过程调用与执行器交互实现覆盖信息和测试样例的记录，执行器则运行多个虚拟机来实现对系统内核持续测试。控制节点通过 Http 和远程控制调用与各计算节点的管理器实施并行数据交互，更新全局数据库信息，突破了传统测试模型对计算资源要求限制和数据竞争的瓶颈。

Li 等[29,35]借助 IDA pro 静态分析工具[134]来识别基本块与基本块之间的跳转，采用一维数组结构来记录跳转访问情况，使用主节点保留全局跳转情况，向各个节点轮询机制来实现更新。主节点的设计在一定程度上降低了通信的开销，具有参考价值。

整体来看，与单机场景下的并行模糊测试相比，多机单原型架构的高可扩展性使得其倍受用户青睐；相较于接下来讨论的多机场景下的多原型架构，多机单原型架构现已孵化出多个开源框架，发展相对成熟。

2.2.4 多机场景下的多原型架构

单机场景到多机场景的演变使并行模糊测试得以扩展至大规模软件系统，单原型到多原型架构的进化使并行模糊测试技术可以在庞大的程序输入空间中高效探索。这些技术的不断发展推动着模糊测试技术高速普及。并行模糊测试研究的最终目标自然是构建一个多机场景下高效、高可用、支持多原型协同的测试框架。近年来，多个著名互联网厂家持之以恒地关注学术界最新研究动向，试图构建出符合上述标准的测试工具。本节将以 CollabFuzz[42]、OSS-Fuzz[36]、OneFuzz[37]为例对现有的多机场景下的多原型架构进行简单介绍。

在 Cupid 的基础上，CollabFuzz 实现了一个协作模糊测试通用框架，着重考虑了如何实现测试用例的高效分发问题，提出了定时调度更新机制、广播式调度同步机制、性能指导的调度更新机制、开销指导的调度更新机制 4 种调度策略。CollabFuzz 架构设计[42]如图 6 所示，框架设计上利用中央调度器负责存储、分析、调度，而模糊测试工具和驱动程序在容器中运行，由驱动程序衔接模糊测试工具与中央调度器的交互。整个框架可以灵活集成分析组件，能够可靠地重现实验环境，并且支持分布式运行，可以轻松扩展到大型集群中。

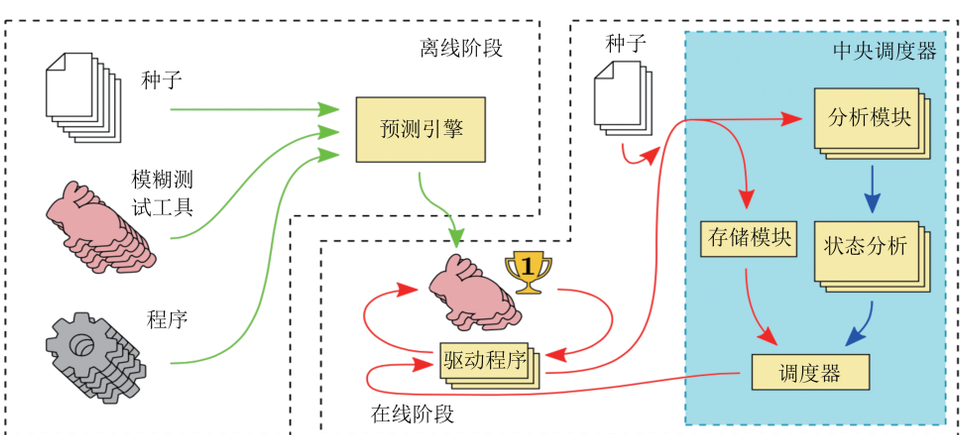

图 6 CollabFuzz 架构设计

Fig. 6 Architecture of CollabFuzz

为了让模糊测试技术更好致力于开源软件的安全维护，谷歌将相关技术集成至 OSS-Fuzz 持续模糊测试平台中，为用户提供可拓展的分布式模糊测试服务。用户将待测项目上传至谷歌云服务之后，OSS-Fuzz 后端将持续对项目进行测试，并分析结果，将漏洞报告交付给用户。OSS-Fuzz 集成了多个著名的模糊测试工具——libFuzzer、AFL＋＋[135]、Honggfuzz[136]。对整个 OSS-Fuzz 平台而言，后端基础架构 ClusterFuzz[51]发挥了举足轻重的作用。ClusterFuzz 提供了全自动的端到端架构，实现了崩溃发现、结果去重、漏洞修复的一体化服务。在并行化工作的设计中，ClusterFuzz 具有高度可扩展性，允许使用者建立任意大小的虚拟机集群并部署在分

布式架构上。依赖于谷歌云服务，ClusterFuzz 建立了中心数据库用于管理测试用例集合。同时，ClusterFuzz 使用了抢占式虚拟机技术进行任务调度。在持续测试过程中，ClusterFuzz 维护中心任务队列以便向下属各虚拟机发放测试任务，分布在各地的抢占式虚拟机将在本地任务完成后主动从中心任务队列中选取下个目标，完成协同工作。遗憾的是，目前 ClusterFuzz 的同步机制粒度主要集中在文件级别，更加高效的同步策略仍有待探索。

Microsoft 在 2020 年发布的 OneFuzz，同样是一个开发者驱动的持续模糊测试平台。其前身为 Microsoft 在 2016 年提出的 Project Springfield[137]。OneFuzz 内置了包括 Honggfuzz，AFL 和 Radamsa 在内的多个模糊测试工具原型，对数个乃至数千个虚拟机调度执行模糊测试任务。在执行流程上，OneFuzz 和 ClusterFuzz 非常类似，多实例间的协同粒度同样停留在种子文件级别。特别地，由于部署阶段高度依赖 Azure 云服务进行集群管理，导致目前 OneFuzz 的可扩展性较弱。可以预计在未来研究中，不管是 ClusterFuzz 还是 OneFuzz 都将持续探索细粒度信息同步的可能性。

现阶段，开源社区正在期待一个高效可行的分布式多原型并行模糊测试架构，以突破当前模糊测试技术的性能瓶颈。虽然诸如 OSS-Fuzz，OneFuzz 等工作实现了兼顾分布式、多原型两大优点，但运转时严重依赖海量计算资源，单核平均性能增长远不能达到用户需求。本文认为这一缺陷是由分布式架构的固有属性决定的——高额的通信开销使得单机模式下有效的信息同步与任务划分算法无一适用于持续模糊测试平台，最终导致众多计算节点只能构建于最基础模糊测试工具之上。因此，多机多原型并行模糊测试目前的关键研究内容应当是节点之间的高效通信机制，在这一基础上，前沿的模糊测试优化技术才有机会扩展至现有的持续模糊测试平台。目前，多机多原型并行模糊测试仍是一个需要长期探索的研究方向。

3　并行化模糊研究发展趋势

近年来，随着多核、众核架构兴起，高性能芯片、云计算技术的不断发展，高计算密度为多计算任务提供有力支撑，并行化成为突破性能瓶颈的关键渠道。尽管并行化模糊测试研究已经取得了一系列立竿见影的成果，但目前仍处于初级阶段，存在许多关键问题有待研究解决。

3.1　多原型并行化模糊测试的效率提升研究

多种相关技术在单核场景下的模糊测试优化中得到应用，控制流分析、数据流分析、符号执行、污点分析等程序分析技术的使用催生了一大批成效卓然的开源工具，数据驱动与智能优化思想也孕育了多个颇受欢迎的模糊测试项目。然而，多原型并行化模糊测试的效率仍有待提高。一方面，基础模糊测试实例的多样性和协同性有待加强。从实例选择上，ENFUZZ 说明基础模糊测试实例的多样性越大，并行化模糊测试性能就越好。不尽如人意的是，大部分并行化方案局限于 3～5 种基础原型，选择原型时依赖简易粗糙的启发式规则。虽然 Cupid[33]、CollabFuzz[42] 就自适应选择原型进行了初步尝试，但也仅是从分支覆盖单维度衡量了 8 种模糊测试工具两两组合时的协同性，涵盖大多数模糊测试工具的同质性与互补性量化分析工作仍虚位以待。从任务分配上，现有分配机制仅考虑到各子任务间低耦合性的需求，虽然较好地避免了冗余重复造成计算资源的浪费，但是忽略了对运行时实际产能和模糊测试工具自身特征的考虑。因此，面向多原型场景下如何依据所选工具特征调整任务分配策略、按需动态调度多种原型工具、实现定制化自适应的计算资源智能分配等问题尚待解决。另一方面，集成架构的可扩展性仍有待完善。传统模糊测试优化领域仍处在蓬勃发展的阶段，未来的研究应该侧重于建立层次清晰、模块分明、协同配合、负载均衡的并行体系，结合模糊测试工具的同质性与互补性评估理论，实现优胜劣汰、与时俱进的模糊测试工具选择机制，保证并行架构下模糊测试的兼容性、灵活性和可扩展性。

3.2　基于并行处理结构的模糊测试加速研究

并行计算具有强架构相关性，现有的并行化模糊测试研究还没有深入到硬件架构这一层级，鲜有考虑软硬协同的一体化发展方向。GPU (graphics processing unit)的众核架构、超长流水线和多线程机制等特性为并行计算奠定了坚实的基础，Eberhardt[138] 率先利用 GPU 的优势来加速单核模糊测试，如何进一步均衡 GPU 的计算

优势和 CPU 的访存效率，充分享用硬件红利实现更高效的并行化模糊测试值得深入探究。近年来，云计算快速发展的引领和驱动作用逐步体现，OSS-Fuzz[36]、OneFuzz[37]、Lian 等[27]就借助云平台的大规模、虚拟化、可伸缩等优势来搭建提供并行化模糊测试云服务，同时也对云环境中网络稳定性、数据安全性等提出了新的要求。目前云环境下的并行方案实现仅侧重于基础模糊测试服务的提供，至于灵活的调度分配机制和可靠性保证机制还有待挖掘探索。

3.3 并行化模糊测试的风险评估和容错研究

在大规模并行模糊测试系统的实际运行中，复杂的调度算法设计、增加的并行节点数量及开放的网络环境带来很多不确定的干扰因素，存在数据竞争、单点故障、传输错误、连接故障、进程阻塞等安全风险。异常情况的发生加大了错误判断的难度，降低了并行模糊测试系统的测试效率，无法保证测试结果的可靠性和准确性。大部分并行模糊测试方案很少考虑到系统的稳定性和安全性，设计人员也缺乏对系统风险的种类、影响因素、危害等的系统认知。因此，如何构建完善的风险评估体系来实现对并行化模糊测试系统脆弱性的量化分析，为大规模并行模糊测试系统的实用化提供标准规范值得深入研究。

虽然有些模糊测试并行架构从宏观角度设计异常处理机制，其控制粒度往往较粗，涵盖的安全风险较为单一。以 ClusterFuzz 为例，中心调度器根据实例崩溃阶段调整其权重，在配置阶段经常宕机的实例会在后续阶段被避免同步[51]。类似地，Lian M 等[27]也针对易发生故障的节点，也提出了基于优先级调度的容错处理方法。但上述方案依赖主控节点的正常运转，无法处理数据竞争、传输错误、进程阻塞等细粒度安全问题。因此，提高系统应对故障的容错能力仍需长期探索，增强系统的容错性、容灾性、可靠性、稳定性是未来并行化模糊测试发展的必然方向。

3.4 并行化模糊测试的评估体系研究

尽管并行化模糊测试方面已经提出了众多方案，但尚未有工作对现有方案的性能优劣给出系统性的评估，主要有以下两方面挑战。其一，缺少统一的并行模糊测试数据集。并行模糊测试近年来受到关注较少，在实验设计的数据集选取阶段没有确定的标准，基本沿用传统模糊测试的数据集[121,139]。具体地，ENFUZZ[25]选择 LAVA-M 数据集[121]和 Google's fuzzer-test-suite[139]作为测试目标，还有工作则在此基础上从真实软件中随机选取若干用以性能测试。现有数据集中大部分测试程序代码逻辑过于简单，运行单个模糊测试进程在有限时间内就能实现充分测试，无法显著区分各并行模糊测试方案的性能差异。其二，缺少统一的评估标准：现有评估标准大多都聚焦在覆盖率和崩溃次数两项指标，很少考虑资源开销、通信成本、系统鲁棒性等多维度的评价。此外，单就覆盖率和崩溃次数两项指标测试上，不同并行工作对于实例运行数量、参与机器数量、测试时间等测试设置各不相同。由于大部分并行工作都没有开源，绝大多数工作在实验时选择与 AFL 原始并行模式对比，UniFuzz[41]选择与运行时间相应倍增后的单核 AFL 模式比较，也有面向内核的并行模糊测试工具 Diskaller[32]选择 Syzkaller[91]作为比较对象展开实验。这样的间接评估很难全面客观地比较不同方案的优劣。因此，亟需建立适合并行模糊测试的通用数据集，制定统一的符合实际应用需求的并行模糊测试测试标准，形成完善的并行模糊测试评估平台。

4 结束语

模糊测试作为软件缺陷检测的关键技术手段，一直是学术界和工业界的研究与实践热点。近年来，随着模糊测试技术的发展、单核计算效率提升的局限、软件功能的丰富与应用场景的拓展，并行化模糊测试成为了颇具前景的新兴领域。本文简要分析了模糊测试的工作流程与局限性，讨论了并行化模糊测试的需求与目标，从应用场景、架构设计、指导信息同步粒度、任务划分粒度等多个维度对现有并行化模糊测试研究进行了梳理，并对已有的多种并行模糊测试方案逐一进行了详尽的介绍，点明了各方案的先进性和局限性，指出了并行化模糊测试的挑战和未来发展方向，为后续研究提供了指导和参考。

参 考 文 献

[1] HABUSHA D. Vulnerability prioritization tops security pros'challenges [EB/OL]. (2020-11-17)[2022-01-04]. https://tinyurl.com/y6os685b.

[2] WANG H, XIE X, LIN S, et al. Locating vulnerabili-

ties in binaries via memory layout recovering[C]// Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Tallinn, Estonia:[s. n.], 2019.

[3] LI J, ZHAO B, ZHANG C. Fuzzing: a survey[J]. Cybersecurity, 2018, 1(1):1-13.

[4] 国家互联网应急中心. 2021 年上半年我国互联网网络安全监测数据分析报告[EB/OL]. (2021-07-31)[2022-01-04]. https://www.cert.org.cn/publish/main/46/2021/20210731090556980286517/20210731090556980286517_.html.
CNCERT. Analysis report of China's Internet network security monitoring data in the first half of 2021[EB/OL]. (2021-07-31)[2022-01-04]. https://www.cert.org.cn/publish/main/46/2021/20210731090556980286517/2021073109055698 0286517 _. html. (in Chinese)

[5] USCYBERCOM. Technicaloutreach division [EB/OL]. (2020-08-10)[2022-01-04]. https://www.cybercom.mil/partnerships-and-outreach/technical.

[6] HELIN A. Radamsa[EB/OL]. [2022-01-04]. https://gitlab.com/akihe/radamsa.

[7] AMINI P, PORTNOY A. Sulley: fuzzing framework [EB/OL]. [2022-01-04]. https://github.com/OpenRCE/sulley.

[8] GODEFROID P, PELEG H, SINGH R. Learn&fuzz: machine learning for input fuzzing[C]//Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). Illinois, USA:[s. n.], 2017.

[9] CUMMINS C, PETOUMENOS P, MURRAY A, et al. Compiler fuzzing through deep learning[C]// Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. Amsterdam, Netherlands:[s. n.], 2018.

[10] YUE T, WANG P, TANG Y, et al. Ecofuzz: adaptive energy-saving greybox fuzzing as a variant of the adversarial multi-armed bandit[C]//Proceedings of the 29th USENIX Security Symposium (USENIX Security 20). [S. l.:s. n.], 2020.

[11] WANG Y, JIA X, LIU Y, et al. Not all coverage measurements are equal: fuzzing by coverage accounting for input prioritization[C]// Proceedings of 2020 Network and Distributed System Security Symposium. San Diego, USA:[s. n.], 2020.

[12] WANG J, SONG C, YIN H. Reinforcement learning-based hierarchical seed for greybox fuzzing[C]// Proceedings of 2021 Network and Distributed System Security Symposium. [S. l.:s. n.], 2021.

[13] SAM H. Zzuf[EB/OL]. [2022-01-04]. https://github.com/samhocevar/zzuf.

[14] MOZILLA S. Funfuzz[EB/OL]. [2022-01-04]. https://github.com/MozillaSecurity/funfuzz.

[15] FAN R, CHANG Y. Machine learning for black-box fuzzing of network protocols[C]//Proceedings of the 19th International Conference on Information and Communications. Beijing, China:[s. n.], 2017.

[16] Peach Tech. Peach-Fuzzer-Community [EB/OL]. [2022-01-04]. https://gitlab.com/peachtech/peach-fuzzer-community.

[17] CADAR C, DUNBAR D, ENGLER D. KLEE: unassisted and automatic generation of high-coverage tests for complexs system programs[C]// Proceedings of USENIX Symposium on Operating Systems Design and Implementation. San Diego, USA:[s. n.], 2008.

[18] GANESH V, LEEK T, RINARD M. Taint-based directed whitebox fuzzing[C]//Proceedings of the 31st International Conference on Software Engineering. Vancouver, Canada:[s. n.], 2009: 474-484.

[19] CHIPOUNOV V, KUZNETSOV V, CANDEA G. S2E: a platform for in-vivo multi-path analysis of software systems[J]. ACM Sigplan Notices, 2011, 46(3): 14.

[20] GODEFROID P, KIEZUN A, LEVIN M Y. Grammar-based whitebox fuzzing[C]//Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation. Tucson, USA:[s. n.], 2008.

[21] BÖHME M, PHAM V T, ROYCHOUDHURY A. Coverage-based greybox fuzzing as markov chain[J]. IEEE Transactions on Software Engineering, 2017, 45(5): 489-506.

[22] CHEN H, XUE Y, LI Y, et al. Hawkeye: towards a desired directed grey-box fuzzer[C]// Proceedings of 2018 ACM SIGSAC Conference on Computer and Communications Security. Toronto, Canada:[s. n.], 2018.

[23] MANÈS V J M, KIM S, CHA S K. Ankou: guiding grey-box fuzzing towards combinatorial difference [C]// Proceedings of the 42nd ACM/IEEE International Conference on Software Engineering. Seoul, Korea:[s. n.], 2020.

[24] 邹燕燕，邹维，尹嘉伟，等. 变异策略感知的并行模糊测试研究[J]. 信息安全学报，2019，5(5):1-16.
ZOU Yanyan, ZOU Wei, YIN Jiawei, et al. Research on mutator strategy-aware parallel fuzzing[J]. Journal of Cyber Security, 2019, 5(5):1-16. (in Chinese)

[25] CHEN Y, JIANG Y, MA F, et al. Enfuzz: ensemble fuzzing with seed synchronization among diverse fuzzers[C]//Proceedings of the 28th USENIX Security Symposium. Santa Clara, USA:[s. n.], 2019.

[26] XU W, KASHYAP S, MIN C, et al. Designing new operating primitives to improve fuzzing performance [C]// Proceedings of 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas, USA:[s. n.], 2017.

[27] 廉美, 邹燕燕, 霍玮, 等. 动态资源感知的并行化模糊测试框架[J]. 计算机应用研究, 2017, 34(1): 52-57.
LIAN Mei, ZOU Yanyan, HUO Wei, et al. Dynamic resource awareness framework for parallel fuzzing[J]. Application Research of Computers, 2017, 34(1): 52-57. (in Chinese)

[28] LIANG J, JIANG Y, CHEN Y, et al. Pafl: extend fuzzing optimizations of single mode to industrial parallel mode[C]// Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Lake Buena Vista, USA:[s. n.], 2018.

[29] LI Y, FENG C, TANG C. A Large-scale parallel fuzzing system[C]// Proceedings of the 2nd International Conference on Advances in ImageProcessing. Chengdu, China:[s. n.], 2018.

[30] YE J, ZHANG B, LI R, et al. Program state sensitive parallel fuzzing for real world software[J]. IEEE Access, 2019(7): 42557-42564.

[31] SONG C, ZHOU X, YIN Q, et al. P-Fuzz: a parallel grey-box fuzzing framework[J]. Applied Sciences, 2019, 9(23): 5100.

[32] 涂序文, 王晓锋, 甘水滔, 等. Diskaller:基于覆盖率制导的操作系统内核漏洞并行挖掘模型[J]. 信息安全学报, 2019, 4(2):69-82.
TU Xuwen, WANG Xiaofeng, GAN Shuitao, et al. Diskaller: kernel vulnerability parallel mining model based on coverage guidance[J]. Journal of Cyber Security, 2019, 4(2):69-82. (in chinese)

[33] GÜLER E, GÖRZ P, GERETTO E, et al. Cupid: automatic fuzzer selection for collaborative fuzzing [C]//Proceedings of 2020 Annual Computer Security Applications Conference. Austin, USA:[s. n.], 2020.

[34] BUTTS RICHÖ. Roving[EB/OL]. (2020-03-02)[2022-01-04]. https://github.com/richo/roving.

[35] LI Y, ZOU H, LIU H. A high efficient technology for parallel fuzzing[C]// Proceedings of the 4th High Performance Computing and Cluster Technologies Conference & the 3rd International Conference on Big Data and Artificial Intelligence. Qingdao, China:[s. n.], 2020.

[36] GOOGLE. OSS-Fuzz: continuous fuzzing for open source software[EB/OL]. [2022-01-04]. https://github.com/google/oss-fuzz.

[37] Project OneFuzz[EB/OL]. (2020-09-15)[2022-01-04]. https://www.microsoft.com/en-us/research/project/project-onefuzz/.

[38] WANG Y, ZHANG Y, PANG C, et al. Facilitating parallel fuzzing with mutually-exclusive task distribution[C]//Proceedings of International Conferece on Security and Privacy in Communication Networks. [S. l.:s. n.], 2021.

[39] PHAM V, NGUYEN M, TA Q. Towards systematic and dynamic task allocation for collaborative parallel fuzzing[C]// Proceedings of the 36th ACM/IEEE International Conference on Automated Software Engineering, New Ideas and Emerging Results. Ann Arbor, USA:[s. n.], 2021.

[40] TANG R, YANG T, FEI Z. A parallel fuzzing method based on two-stage mutation[C]//Proceedings of 2021 International Conference on Green Communication, Network, and Internet of Things. Kunming, China:[s. n.], 2021.

[41] ZHOU X, WANG P, LIU C, et al. Unifuzz: optimizing distributed fuzzing via dynamic centralized task scheduling[EB/OL]. [2022-01-04]. https://arxiv.org/abs/2009.06124.

[42] ÖSTERLUND S, GERETTO E, JEMMETT A, et al. Collabfuzz: a framework for collaborative fuzzing [C]// Proceedings of the 14th European Workshop on Systems Security. [S. l.:s. n.], 2021.

[43] MILLER B P, FREDRIKSEN L, SO B. An empirical study of the reliability of UNIX utilities[J]. Communications of the ACM, 1990, 33(12): 32-44.

[44] PWN2OWN. The perfect antidote to fanboys who say their platform is safe ars technica[EB/OL]. (2014-03-14)[2022-01-04]. https://arstechnica.com/information-technology/2014/03/pwn2own-the-perfect-antidote-to-fanboys-who-say-their-platform-is-safe/.

[45] JACK K. Charlie miller reveals his process for security research[EB/OL]. (2011-03-14)[2022-01-04]. https://resources.infosecinstitute.com/topic/how-charlie-miller-does-research/.

[46] STEPHENS N, GROSEN J, SALLS C, et al. Driller: augmenting fuzzing through selective symbolic execution[C]// Proceedings of 2016 Network and Distributed System Security Symposium. San Diego, USA:[s. n.], 2016.

[47] BUG FINDER. I am a mayhem, the hacking machine that won DARPA's cyber grand challenge. AMA![Z/OL]. (2016-08-11)[2022-01-04]. www.reddit.com/r/IAmA/comments/4x9yn3/iama_mayhem_the_hacking_machine_that_won_darpas/.

[48] RIZZI E. The cyber grand challenge[EB/OL]. (2016-11-26)[2022-01-04]. https://blogs.grammatech.com/the-cyber-grand-challenge.

[49] FOSTER B. Binspector: evolving a security tool[EB/OL]. (2015-05-28)[2022-01-04]. http://web.archive.org/web/20181020154300/.

[50] NEFF M. Tutorial: mutiny fuzzing framework and decept proxy[EB/OL]. (2018-01-03)[2022-01-04]. http://blog.talosintelligence.com/2018/01/tutorial-mutiny-fuzzing-framework-and.html.

[51] Chrome Security Team. ClusterFuzz[EB/OL].[2022-01-04]. https://google.github.io/clusterfuzz/.

[52] Chrome Security Team. Security bugs[EB/OL].[2022-01-04]. https://www.chromium.org/Home/chromium-security/bugs.

[53] AIZATSKY M, SEREBRYANY K, CHANG O, et al. Announcing OSS-FUZZ: continuous fuzzing for open source software[EB/OL]. (2016-12-01)[2022-01-04]. https://opensource.googleblog.com/2016/12/announcing-oss-fuzz-continuous-fuzzing.html.

[54] BOUNIMOVA E, GODEFROID P, MOLNAR D. Billions and billions of constraints: whitebox fuzz testing in production[C]//Proceedings of the 35th International Conference on Software Engineering. San Francisco, USA:[s.n.], 2013.

[55] Virtualization Security Team. Fuzzing para-virtualized devices in hyper-v-microsoft security response center[EB/OL]. (2019-01-28)[2022-01-04]. https://msrc-blog.microsoft.com/2019/01/28/fuzzing-para-virtualized-devices-in-hyper-v/.

[56] American fuzzy lop[EB/OL].[2022-01-04]. https://lcamtuf.coredump.cx/afl/.

[57] WEN C, WANG H, LI Y, et al. Memlock: memory usage guided fuzzing[C]// Proceedings of the 42nd ACM/IEEE International Conference on Software Engineering. Seoul, Korea:[s.n.], 2020.

[58] WANG H, XIE X, LI Y, et al. Typestate-guided fuzzer for discovering use-after-free vulnerabilities[C]// Proceedings of the 42nd ACM/IEEE International Conference on Software Engineering. Seoul, Korea:[s.n.], 2020.

[59] ÖSTERLUND S, RAZAVI K, BOS H, et al. ParmeSan: sanitizer-guided greybox fuzzing[C]//Proceedings of the 29th USENIX Security Symposium. Boston, USA:[s.n.], 2020.

[60] PETSIOS T, ZHAO J, KEROMYTIS A D, et al. Slowfuzz: automated domain-independent detection of algorithmic complexity vulnerabilities[C]// Proceedings of 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas, USA:[s.n.], 2017.

[61] DING R, KIM Y, SANG F, et al. Hardware support to improve fuzzing performance and precision[C]// Proceedings of 2021 ACM SIGSAC Conference on Computer and Communications Security. [S.l.:s.n.], 2021.

[62] NAGY S, NGUYEN-TUONG A, HISER J D, et al. Breaking through binaries: compiler-quality instrumentation for better binary-only fuzzing[C]//Proceedings of the 30th USENIX Security Symposium. Vancouver, Canada:[s.n.], 2021.

[63] SIMON L, VERMA A. Improving fuzzing through controlled compilation[C]//Proceedings of 2020 IEEE European Symposium on Security and Privacy. [S.l.:s.n.], 2020.

[64] NAGY S, HICKS M. Full-speed fuzzing: reducing fuzzing overhead through coverage-guided tracing[C]//Proceedings of 2019 IEEE Symposium on Security and Privacy. San Francisco, USA:[s.n.], 2019.

[65] 高凤娟，王豫，司徒凌云，等. 基于深度学习的混合模糊测试方法[J]. 软件学报，2021，32(4)：988-1005.
GAO Fengjuan, WANG Yu, SITU Lingyun, et al. Deep learning-based hybrid fuzz testing[J]. Journal of Software, 2021, 32(4): 988-1005. (in Chinese)

[66] LIN P, HONG Z, LI Y, et al. A priority based path searching method for improving hybrid fuzzing[J]. Computers & Security, 2021, 105: 102242.

[67] KIM K, JEONG D, KIM C H, et al. HFL: hybrid-fuzzing on the linux kernel[C]// Proceedings of 2020 Network and Distributed System Security Symposium. San Diego, USA:[s.n.], 2020.

[68] BORZACCHIELLO L, COPPA E, DEMETRESCU C. FUZZOLIC: mixing fuzzing and concolic execution[J]. Computers & Security, 2021, 108: 102368.

[69] YUN I, LEE S, XU M, et al. QSYM: a practical concolic execution engine tailored for hybrid fuzzing[C]//Proceedings of the 27th USENIX Security Symposium. Baltimore, USA:[s.n.], 2018.

[70] CUI B, WANG F, HAO Y, et al. WhirlingFuzzwork: a taint-analysis-based API in-memory fuzzing framework[J]. Soft Computing, 2017, 21(12): 3401-3414.

[71] GAN S, ZHANG C, CHEN P, et al. GREYONE: data flow sensitive fuzzing [C]//Proceedings of the 29th USENIX Security Symposium. Boston, USA: [s. n.], 2020.

[72] CHEN P, CHEN H. Angora: efficient fuzzing by principled search[C]//Proceedings of 2018 IEEE Symposium on Security and Privacy. San Francisco, USA: [s. n.], 2018.

[73] ASCHERMANN C, SCHUMILO S, BLAZYTKO T, et al. REDQUEEN: fuzzing with input-to-state correspondence [C]// Proceedings of 2019 Network and Distributed System Security Symposium. San Diego, USA: [s. n.], 2019.

[74] ZONG P, LV T, WANG D, et al. FuzzGuard: filtering out unreachable inputs in directed grey-box fuzzing through deep learning [C]//Proceedings of the 29th USENIX Security Symposium. [S. l.: s. n.], 2020.

[75] ZAKERI NASRABADI M, PARSA S, KALAEE A. Format-aware learn&fuzz: deep test data generation for efficient fuzzing[J]. Neural Computing and Applications, 2021, 33(5): 1497-1513.

[76] SHE D, PEI K, EPSTEIN D, et al. Neuzz: efficient fuzzing with neural program smoothing[C]//Proceedings of 2019 IEEE Symposium on Security and Privacy. San Francisco, USA: [s. n.], 2019.

[77] LYU C, JI S, ZHANG C, et al. MOPT: optimized mutation scheduling for fuzzers [C]//Proceedings of the 28th USENIX Security Symposium. Santa Clara, USA: [s. n.], 2019.

[78] LI Z, ZHAO H, SHI J, et al. An intelligent fuzzing data generation method based on deep adversarial learning[J]. IEEE Access, 2019(7): 49327-49340.

[79] LI Y, JI S, CHEN Y, et al. Unifuzz: a holistic and pragmatic metrics-driven platform for evaluating fuzzers [C]//Proceedings of the 30th USENIX Security Symposium. [S. l.: s. n.], 2021.

[80] WANG M, LIANG J, ZHOU C, et al. Industrial oriented evaluation of fuzzing techniques [C]//Proceedings of the 14th IEEE Conference on Software Testing, Verification and Validation. Cleveland, USA: [s. n.], 2021.

[81] HAZIMEH A, HERRERA A, PAYER M. Magma: a ground-truth fuzzing benchmark[C]. Proceedings of ACM on Measurement and Analysis of Computing Systems. [S. l.: s. n.], 2020: 1-29.

[82] ZHU X, FENG X, JIAO T, et al. A feature-oriented corpus for understanding, evaluating and improving fuzz testing[C]// Proceedings of 2019 ACM Asia Conference on Computer and Communications Security. Auckland, New Zealand: [s. n.], 2019.

[83] WANG J, DUAN Y, SONG W, et al. Be sensitive and collaborative: analyzing impact of coverage metrics in greybox fuzzing[C]//Proceedings of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses. Beijing, China: [s. n.], 2019.

[84] KLEES G, RUEF A, COOPER B, et al. Evaluating fuzz testing[C]// Proceedings of 2018 ACM SIGSAC Conference on Computer and Communications Security. Toronto, Canada: [s. n.], 2018.

[85] ZHU X, BÖHME M. Regression greybox fuzzing [C]// Proceedings of 2021 ACM SIGSAC Conference on Computer and Communications Security. [S. l.: s. n.], 2021.

[86] DAI J, ZHANG Y, XU H, et al. Facilitating vulnerability assessment through PoC migration [C]// Proceedings of 2021 ACM SIGSAC Conference on Computer and Communications Security. [S. l.: s. n.], 2021.

[87] SUN H, SHEN Y, WANG C, et al. Healer: relation learning guided kernel fuzzing[C]// Proceedings of the 28th ACM SIGOPS Symposium on Operating Systems Principles. [S. l.: s. n.], 2021.

[88] GONG S, ALTINBÜKEN D, FONSECA P, et al. Snowboard: finding kernel concurrency bugs through systematic inter-thread communication analysis [C]// Proceedings of the 28th ACM SIGOPS Symposium on Operating Systems Principles. [S. l.: s. n.], 2021.

[89] KIM S, XU M, KASHYAP S, et al. Finding bugs in file systems with an extensible fuzzing framework[J]. ACM Transactions on Storage (TOS), 2020, 16(2): 1-35.

[90] JEONG D R, KIM K, SHIVAKUMAR B, et al. Razzer: finding kernel race bugs through fuzzing [C]// Proceedings of 2019 IEEE Symposium on Security and Privacy. San Francisco, USA: [s. n.], 2019.

[91] SYZKALLER. Kernel fuzzer [EB/OL]. [2022-01-04]. https://github.com/google/syzkaller.

[92] PAILOOR S, ADAY A, JANA S. MoonShine: optimizing os fuzzer seed selection with trace distillation [C]//Proceedings of the 27th USENIX Security Symposium. Baltimore, USA: [s. n.], 2018.

[93] CHOI J, KIM K, LEE D, et al. NtFuzz: enabling type-aware kernel fuzzing on windows with static binary analysis[C]//Proceedings of 2021 IEEE Symposium on Security and Privacy. [S. l.: s. n.], 2021.

[94] PAN G, LIN X, ZHANG X, et al. V-Shuttle: scalable and semantics-aware hypervisor virtual device fuzz-

ing[C]// Proceedings of 2021 ACM SIGSAC Conference on Computer and Communications Security. [S. l. :s. n.], 2021.

[95] XU H, QIN G, ZHU J, et al. Framework for state-aware virtual hardware fuzzing[J]. Wireless Communications and Mobile Computing, 2021(5):1-14.

[96] LI X, WU Z, WEI Q, et al. Uisfuzz: an efficient fuzzing method for cpu undocumented instruction searching[J]. IEEE Access, 2019(7): 149224-149236.

[97] SCHUMILO S, ASCHERMANN C, ABBASI A, et al. Hyper-cube: high-dimensional hypervisor fuzzing [C]// Proceedings of 2020 Network and Distributed System Security Symposium. San Diego, USA:[s. n.], 2020.

[98] KIM J, YU J, KIM H, et al. Firm-cov: high-coverage greybox fuzzing for IoT firmware via optimized process emulation[J]. IEEE Access, 2021(9): 101627-101642.

[99] GUI Z, SHU H, KANG F, et al. Firmcorn: vulnerability-oriented fuzzing of iot firmware via optimized virtual execution[J]. IEEE Access, 2020(8): 29826-29841.

[100] LV W, XIONG J, SHI J, et al. A deep convolution generative adversarial networks based fuzzing framework for industry control protocols[J]. Journal of Intelligent Manufacturing, 2021, 32(2): 441-457.

[101] KIM S J, CHO J, LEE C, et al. Smart seed selection-based effective black box fuzzing for IIoT protocol[J]. The Journal of Supercomputing, 2020, 76(12): 10140-10154.

[102] CHOI J, GRIECO G, KIM D, et al. Smartian: enhancing smart contract fuzzing with static and dynamic data-flow analyses[C]// Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering. Melbourne, Australia: [s. n.], 2021.

[103] NGUYEN T D, PHAM L H, SUN J, et al. Sfuzz: an efficient adaptive fuzzer for solidity smart contracts[C]//Proceedings of the 42nd International Conference on Software Engineering. Seoul, Korea: [s. n.], 2020.

[104] HE J, BALUNOVIC M, AMBROLADZE N, et al. Learning to fuzz from symbolic execution with application to smart contracts[C]// Proceedings of 2019 ACM SIGSAC Conference on Computer and Communications Security. London, UK:[s. n.], 2019.

[105] ASHRAF I, MA X, JIANG B, et al. GasFuzzer: fuzzing ethereum smart contract binaries to expose gas-oriented exception security vulnerabilities[J]. IEEE Access, 2020(8):99552-99564.

[106] WANG M, WU Z, XU X, et al. Industry practice of coverage-guided enterprise-level DBMS fuzzing[C]// Proceedings of the 43rd International Conference on Software Engineering. Madrid, Spain:[s. n.], 2021.

[107] ZHONG R, CHEN Y, HU H, et al. Squirrel: testing database management systems with language validity and coverage feedback[C]// Proceedings of 2020 ACM SIGSAC Conference on Computer and Communications Security. [S. l. :s. n.], 2020.

[108] HE X, XIE X, LI Y, et al. Sofi: reflection-augmented fuzzing for javascript engines[C]// Proceedings of 2021 ACM SIGSAC Conference on Computer and Communications Security. [S. l. :s. n.], 2021.

[109] XU W, PARK S, KIM T. Freedom: engineering a state-of-the-art DOM fuzzer[C]// Proceedings of 2020 ACM SIGSAC Conference on Computer and Communications Security. [S. l. :s. n.], 2020.

[110] PARK S, XU W, YUN I, et al. Fuzzing javascript engines with aspect-preserving mutation[C]//Proceedings of 2020 IEEE Symposium on Security and Privacy. [S. l. :s. n.], 2020.

[111] HAN H S, OH D H, CHA S K. CodeAlchemist: semantics-aware code generation to find vulnerabilities in javascript engines[C]// Proceedings of 2019 Network and Distributed System Security Symposium. San Diego, USA:[s. n.], 2019.

[112] CHEN Y, ZHONG R, HU H, et al. One engine to fuzz'em all: generic language processor testing with semantic validation[C]//Proceedings of 2021 IEEE Symposium on Security and Privacy. [S. l. :s. n.], 2021.

[113] YAO P, HUANG H, TANG W, et al. Fuzzing SMT solvers via two-dimensional input space exploration[C]// Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. [S. l. :s. n.], 2021.

[114] MARCOZZI M, TANG Q, DONALDSON A F, et al. Compiler fuzzing: how much does it matter? [C]//Proceedings of ACM on Programming Languages. [S. l. :s. n.], 2019:1-29.

[115] LUO W, CHAI D, RUN X, et al. Graph-based fuzz testing for deep learning inference engines[C]//Proceedings of the 43rd International Conference on Software Engineering. Madrid, Spain:[s. n.], 2021.

[116] GAO X, SAHA R K, PRASAD M R, et al. Fuzz testing based data augmentation to improve robustness of deep neural networks[C]//Proceedings of the

42nd International Conference on Software Engineering. Seoul, Korea:[s. n.], 2020.

[117] ODENA A, OLSSON C, ANDERSEN D, et al. Tensorfuzz: debugging neural networks with coverage-guided fuzzing[C]// Proceedings of the 36th International Conference on Machine Learning. Long Beach, USA:[s. n.], 2019.

[118] GODEFROID P, HUANG B Y, POLISHCHUK M. Rest api data fuzzing[C]//Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software. [S. l.:s. n.], 2020.

[119] ATLIDAKIS V, GODEFROID P, POLISHCHUK M. Restler: stateful rest api fuzzing[C]//Proceedings of the 41st International Conference on Software Engineering. Montreal, Canada:[s. n.], 2019.

[120] VAN STEEN M, TANENBAUM A. Distributed systems principles and paradigms[M]. Hoboken: Pearson Prentice Hall, 2006.

[121] DOLAN-GAVITT B, HULIN P, KIRDA E, et al. Lava: large-scale automated vulnerability addition [C]//Proceedings of 2016 IEEE Symposium on Security and Privacy. San Jose, USA:[s. n.], 2016.

[122] YOU W, WANG X, MA S, et al. Profuzzer: on-the-fly input type probing for better zero-day vulnerability detection[C]//Proceedings of 2019 IEEE Symposium on Security and Privacy. San Francisco, USA:[s. n.], 2019.

[123] XIE Y. Using grid computing for large scale fuzzing [D]. Lisboa: University of Lisbon, 2010.

[124] 王连赢. 文件触发类二进制程序漏洞挖掘技术研究[D]. 北京:北京邮电大学, 2015.
WANG Lianying. Research on binary-executable-oriented file format vulnerability detection[D]. Beijing: Beijing University of Posts and Telecommunications, 2015. (in Chinese)

[125] LEE W H, SRIRANGAM R M, KRISHNAN S P T. On designing an efficient distributed black-box fuzzing system for mobile devices[C]// Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security. Singapore:[s. n.], 2015.

[126] BETERKE F. Distributed evolutionary fuzzing with evofuzz[EB/OL]. [2022-01-04]. http://dl. gi. de/handle/20. 500. 12116/878.

[127] STAATS M, PĂSĂREANU C. Parallel symbolic execution for structural test generation[C]// Proceedings of the 19th International Symposium on Software Testing and Analysis. Trento, Italy:[s. n.], 2010.

[128] CIORTEA L, ZAMFIR C, BUCUR S, et al. Cloud9: a software testing service[J]. Acm Sigops Operating Systems Review, 2010, 43(4): 5-10.

[129] GODEFROID P, LEVIN M Y, MOLNAR D. Sage: whitebox fuzzing for security testing[J]. Communications of the ACM, 2012, 55(3): 40-44.

[130] 梁洪亮, 阳晓宇, 董钰, 等. 并行化智能模糊测试[J]. 清华大学学报(自然科学版), 2014, 54(1): 14-19.
LIANG Hongliang, YANG Xiaoyu, DONG Yu, et al. Parallel smart fuzzing test[J]. Journal of Tsinghua University(Science and Technology), 2014, 54(1), 14-19. (in Chinese)

[131] Defensics Fuzz Testing. Tool & Services | Synopsys [EB/OL]. [2022-01-04]. https://www. synopsys. com/software-integrity/security-testing/fuzz-testing. html.

[132] LEMIEUX C, SEN K. Fairfuzz: a targeted mutation strategy for increasing greybox fuzz testing coverage [C]// Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. Montpellier, France:[s. n.], 2018.

[133] LIBFUZZER. A library for coverage-guided fuzz testing[EB/OL]. [2022-01-04]. https://llvm. org/docs/Lib Fuzzer. html.

[134] IDA Pro. Hex-rays[EB/OL]. [2022-01-04]. https://hex-rays. com/ida-pro/.

[135] FIORALDI A, MAIER D, EIßFELDT H, et al. AFL++ : combining incremental steps of fuzzing research[C]//Proceedings of the 14th USENIX Workshop on Offensive Technologies. [S. l.:s. n.], 2020.

[136] Honggfuzz[EB/OL]. [2022-01-04]. https://github. com/google/honggfuzz.

[137] Project Springfield: a cloud service built entirely in F#[EB/OL]. (2016-12-13)[2022-01-04]. https://devblogs. microsoft. com/dotnet/project-springfield-a-cloud-service-built-entirely-in-f/.

[138] RYAN EBERHARDT. Let's build a high-performance fuzzer with GPUs! | Trail of bits blog[EB/OL]. [2022-03-23]. https://blog. trailofbits. com/2020/10/22/lets-build-a-high-performance-fuzzer-with-gpus/.

[139] Google/Fuzzer-Test-Suite: set of tests for fuzzing engines[EB/OL]. [2022-03-23]. https://github. com/google/fuzzer-test-suite.

责任编辑 董 莉