计算机科学
COMPUTER SCIENCE

多线程数据竞争检测技术研究综述

赵静文，付岩，吴艳霞，陈俊文，冯云，董继斌，刘嘉琪

引用本文

赵静文，付岩，吴艳霞，陈俊文，冯云，董继斌，刘嘉琪. 多线程数据竞争检测技术研究综述[J]. 计算机科学, 2022, 49(6): 89-98.

ZHAO Jing-wen, FU Yan, WU Yan-xia, CHEN Jun-wen, FENG Yun, DONG Ji-bin, LIU Jia-qi. Survey on Multithreaded Data Race Detection Techniques[J]. Computer Science, 2022, 49(6): 89-98.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210700187

多线程数据竞争检测技术研究综述

赵静文[1] 付 岩[1] 吴艳霞[1] 陈俊文[2] 冯 云[2] 董继斌[1] 刘嘉琪[1]
1 哈尔滨工程大学计算机科学与技术学院 哈尔滨 150001
2 北京控制与电子技术研究所 北京 100038
(zhaojw123@hrbeu.edu.cn)

摘 要 随着多核处理器在现代计算机设备中的流行,在软件中使用多线程程序的频率也随之增加。但多线程程序的不确定性会导致程序在运行过程中出现数据竞争、原子性违背、顺序违背和死锁等并发问题。研究发现,在所有并发缺陷中,数据竞争所占的比例最大,而且大多数原子性违背和顺序违背也是由数据竞争引起的。为解决这一问题,学者们先后提出了相关的检测技术,文中对近年来该领域的研究技术进行了总结。首先,介绍了数据竞争的相关概念和产生原因,以及数据竞争检测的主要思想;然后根据程序是否执行将现有的数据竞争检测技术分为静态分析、动态分析和混合检测技术三大类,归纳分析了每类技术的特点并进行了详细的比较;随后,从程序员角度阐明了现有检测技术存在的问题;最后,根据发展现状,对该领域的未来发展方向进行了分析与探讨。

关键词:数据竞争;并发缺陷;检测技术;静态分析;动态分析
中图法分类号 TP311

Survey on Multithreaded Data Race Detection Techniques

ZHAO Jing-wen[1],FU Yan[1],WU Yan-xia[1],CHEN Jun-wen[2],FENG Yun[2],DONG Ji-bin[1] and LIU Jia-qi[1]
1 College of Computer Science and Technology,Harbin Engineering University,Harbin 150001,China
2 Beijing Institute of Control and Electronic Technology,Beijing 100038,China

Abstract Nowadays the multi-core processors and threaded parallel programs are increasingly more used. However,the uncertainty of multi-threaded program leads to concurrency problems such as data race,atomicity violation,order violation and deadlock in the process of program running. Recent researches show that data race accounts for the largest proportion of concurrency bug,and most atomicity violation and order violation are caused by data race. This paper summarizes the related detection techniques in recent years. Firstly,the related concepts,causes,and the main ideas of data race detection are introduced. Then,the existing data race detection techniques in multi-threaded program are classified into three types:static analysis,dynamic analysis and hybrid detection techniques,and their characteristics are summarized comprehensively and compared in detail. Next,the limitations of existing data race detection tools are discussed. Finally,future research directions and challenges in this field are discussed.

Keywords Data race,Concurrency bug,Detection techniques,Static analysis,Dynamic analysis

1 引言

随着计算机的发展,多线程程序已被广泛应用于软件系统的开发过程中。多核CPU的应用大幅提升了程序运行效率,但多线程程序存在巨大的并发执行交织空间,使得程序员在并发编程的过程中容易忽略执行交错而导致的数据竞争等并发缺陷,一些并发缺陷若不能被及时发现,则会对程序之后的运行造成很大的影响。因此,如何对并发缺陷进行有效检测一直是研究的热点。

多线程中的这些并发问题会使程序陷入一种不确定状态,多线程程序运行的随机性导致程序每次执行的结果可能会不一致。在对程序进行测试时,一些错误不容易被发现,但如果运用到生产生活中,一旦有一次运行出现问题,后果将不堪设想[1]。在现实世界中,大多数服务器和高端关键软件都是多线程的,这些并发错误已经导致了历史上一些与计算机相关的严重事故,例如:纳斯达克OMX系统发生并发错误故障造成了1300万美元的损失;2004年,数据竞争导致北美洲数千万人遭遇停电事故[2];Therac-25放射治疗仪故障致使5人失去生命[3]等。

由于多线程之间数据是共享的,因此在程序运行过程中会出现多个线程在同一时间对同一资源进行读写访问,如果至少有一个访问为写[4],最终会导致不同的程序运行结果。一般而言,并发缺陷具有4个方面的特点[5]:1)难以重现,线程执行交织依赖于调度程序,并发程序的运行具有不确定性,仅在特定的、随机的执行交织下才会暴露;2)难以调试,并发缺陷从出现到暴露具有一定的时间差;3)难以处理,随着线程

到稿日期:2021-07-19 返修日期:2021-12-11
通信作者:付岩(Fuyan@hrbeu.edu.cn)

数量和程序大小的增加，并发程序之间的交织空间呈指数级扩展，分析并发程序在计算上变得困难；4）难以修复，一些缺陷虽然已被检测到，但对其进行修复可能会导致新的错误。近年来，越来越多的研究人员开始关注数据竞争等并发缺陷问题，并提出了许多有效的数据竞争检测技术，从宏观上看，主要包括静态分析、动态分析和混合检测这3种方法。

本文旨在对CPU并行程序中数据竞争检测的相关工作进行总结和概括，主要从检测思想和检测技术两个角度对数据竞争的检测和相关研究进行讨论，主要思想包括Happens-before关系、锁集法和混合法，并对检测技术进行了特点归纳和详细比较，通过多角度分析数据竞争检测技术的研究现状和其中存在的问题，探讨了未来的研究方向。

2 相关背景

多线程程序中常见的并发缺陷包括但不限于数据竞争、原子性违背、顺序违背和死锁四大类。研究发现，在所有并发缺陷中，数据竞争所占比例最大，并且在大部分情况下，数据竞争也是原子性违背和顺序违背产生的本质原因。因此本文主要对数据竞争这种并发缺陷展开研究，总结其相关思想和检测技术。

2.1 数据竞争简介

定义1 数据竞争：在多线程程序中，两个或两个以上线程对同一共享内存单元进行访问，且至少一个访存操作为写操作，同时这些操作没有被锁保护或者没有被相同的锁保护[6]。图1给出了一个简单的数据竞争程序。

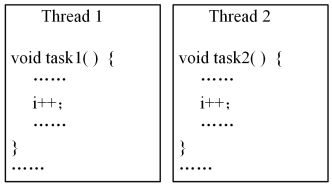

图1 一个简单的数据竞争示例

Fig. 1 Simple example of data race

如图2所示，共享变量i的初始值为0，现有两个线程对变量执行+1的操作，理论上编程人员期望两个线程分别执行$i=i+1$。但实际CPU操作过程并非如此，编译器会将$i=i+1$拆分成3个操作：1）i被CPU从内存中拷贝到缓存；2）对i执行+1；3）把i写回内存中。

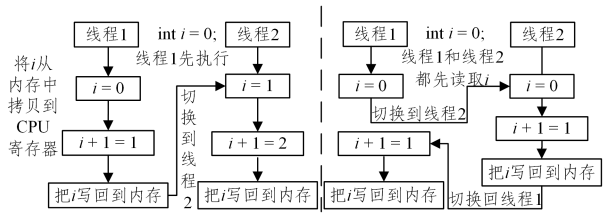

图2 两个线程造成数据竞争模式

Fig. 2 Two threads cause data race

两个线程可能会同时对i进行操作，由于线程的交错执行，在对i执行操作时会产生问题，若CPU在执行上述3个操作中的任何一个操作时切换线程，就可能会导致最终的执行结果和预想结果不一致。如果线程1先读取i的值，执行++操作，随后线程2读取i的值，执行++操作时，i的值会正确地增加2；但如果线程1和线程2都先读取了i的值，然后分别执行++操作，那么i的值会错误地只增加一次，线程1或者线程2对i的一个更新丢失。由于程序没有同步机制保护，因此在运行过程中每种情况都有可能发生，造成数据竞争。

2.2 主要思想

在数据竞争缺陷检测中，主要采用了以下3种基本思想：先发生于法（Happens-before）、锁集法（Lockset）和混合法（Hybrid）。

2.2.1 先发生于法

Happens-before的概念最初是由Lamport[7]提出的，其针对并发缺陷的可见性问题，使用Happens-before关系来约束两个操作之间的执行顺序：1）在同一个线程中需要满足代码顺序执行；2）在线程间要保证正确同步的执行顺序。

而基于Happens-before关系的数据竞争判断方法是当两个或多个线程访问同一个共享变量，并且不满足上述两个原则中的任何一个时，报告潜在的数据竞争，表明该变量没有得到适当的保护，即两个访存操作不在同一个线程中：

$$isSameThread(V_n, V_m) == \text{false} \tag{1}$$

并且不满足Happens-before关系，即：

$$\neg(V_n \prec V_m) \vee \neg(V_m \prec V_n) \tag{2}$$

Happens-before思想理论上没有误报，但其高度依赖于调度程序产生的交错，对线程交错敏感，会产生漏报，并且由于Happens-before关系的方法涉及所有同步操作和共享变量的读取写入操作，导致开销较大，可扩展性较差。

2.2.2 锁集法

锁集法检查两个线程是否在持有公共锁的同时访问共享变量。对于每个共享内存地址，该算法维护一个候选锁集。通常开发人员在编写程序时使用锁操作对临界区进行保护，共享变量的访存操作一般都包含在临界区内，在程序的执行过程中，共享变量和其相关的锁集会发生相应的变化。若某一共享变量在多线程程序运行期间被一个或者多个锁保护，那么就断定该共享变量所涉及的操作不会发生数据竞争；反之，则有可能出现数据竞争。

图3给出了锁集法的操作步骤。候选锁集初始化为mutex1和mutex2，当线程启动获取锁mutex1后，锁集合从空集变为包含mutex1的集合。当共享变量i被访问时，候选锁集与锁集合相交。随后，在使用mutex2访问i时，候选锁集与锁集合再次相交，由于候选锁集仅包含mutex1，两者之间没有公共锁，因此会成为一个空集，发出警告。

	锁定集合	候选锁集
	{ }	{mutex1,mutex2}
1 mutex mutex1,mutex2;		
2 mutex_lock(&mutex1);	{mutex1}	
3 i = i + 1;		{mutex1}
4 mutex_ulock(&mutex1);	{ }	
5 mutex_lock(&mutex2);	{mutex2}	
6 i = i + 1;		{ }
7 mutex_ulock(&mutex2);	{ }	

图3 锁集算法的操作步骤

Fig. 3 Operation steps of lockset algorithm

对于任意两个访存操作 S_n 和 S_m，用 $IsLock(S_n, S_m)$ 表示两个访存操作是否由同一个锁信息保护[8]。但是使用这种思想会出现很多误报。在检测时会忽略除锁之外的所有同步操作，导致检测到的一些数据竞争并不是真正的竞争。

2.2.3 混合法

单独使用 Happens-before 方法会导致较高的漏报率；单独使用锁集法时，由于它不考虑除锁之外的其他一些同步操作，因此也会产生大量的误报，但是该方法复杂度较低，可扩展性较强，一些研究人员把这两种思想相结合，提出了混合法。混合算法的思想主要是以下两种：

(1)先通过 Happens-before 关系找到所有可能的并发操作，而后使用 lockset 算法一一检验有无公共锁对程序进行保护；

(2)先通过 lockset 算法找到没有被公共锁集保护的可疑数据竞争，随后应用 Happens-before 关系判断这些竞争的访问操作之间有无准确的先后关系。

混合法弥补了两种算法的缺陷，在实际使用中取得了较好的检测结果，误报率和漏报率都较低。但由于使用了多种算法，复杂度相对较高。

3 数据竞争检测技术

识别并发错误非常耗时，原因是，数据竞争分为良性数据竞争(对程序结果没有影响，如生产者消费者模式)和有害数据竞争，而传统的检测很少区分出真正有害的数据竞争，准确率较低，导致程序员需要花费大量时间进行验证，且通常需要程序员有丰富的调试经验。研究人员在数据竞争检测方面做了大量研究，逐步降低假阳性和假阴性，提高检测准确率，协助程序员及时准确地发现错误。如图 4 给出了数据竞争检测过程，按照程序是否运行和使用平台的不同，将检测手段分为静态分析、动态分析和混合检测 3 类。

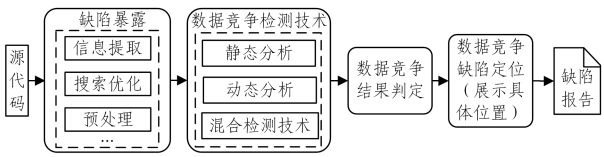

图 4 数据竞争检测过程

Fig. 4 Data race detection process

3.1 静态分析技术

静态分析不需要实际执行程序就能够检测出在实际执行中难以到达路径上的并发错误。数据竞争静态分析没有执行时的动态开销，通常具有较高的可靠性和较低的漏报率，但是需要较多的人工干涉，误报率较高。本节总结了用于检测数据竞争的主要静态分析技术，主要从数据流分析、锁集法、混合法和模型检测 4 个方面展开。

数据流分析技术[9]直接对原始程序进行分析，通过分析共享变量的信息进行检测。RacerX[10]采用流敏感的过程间分析方法，并通过一种错误严重性估计机制来减少误报，但无法保证所有竞争都能被检测到。Chord[11]通过二进制决策图对程序进行一系列处理，以缩小潜在的候选种族范围。针对中断驱动程序，Wu 等[12]提出了一种基于数据流的面向顺序化后过程间分析方法，增加了中断驱动程序顺序化后的可扩展性。为了提高检测的准确度，Zhang 等[13]提出了一种对上下文敏感的数据竞争检测工具 Conracer，使用控制流分析和转义分析构造程序的调用图和转义对象，有效减少了假阳性和假阴性，但执行时间较长。

锁集法检查两个线程在访问时是否持有公共锁。Relay[14]通过静态分析程序上每一个点的锁集来判定是否发生数据竞争，采用相对锁集的概念，按照自底向上的函数调用方法对每一个函数进行锁集计算，在相对锁集计算完毕的同时也生成了相应的共享变量读写等访存信息。此方法可以分析大型程序代码，并且一般没有漏报，但采用非常保守的方式进行别名分析，存在较多误报。Locksmith[15]采用抽象数据流图以及标签流约束，将共享变量的锁集关系展开分析，得到数据竞争。Andrianov 等[16]在锁集算法上简化内存模型，削减由于资源和多个错误之间不平衡造成的误报。对于中断驱动程序中的竞争，Chopra 等[17]对内核 API 库及使用的同步机制进行建模，研究了一种基于锁集的高效分析技术。

通过有选择的监测内存操作，采样方法可以降低锁集算法的执行开销。Racez[18]以准确性换复杂性，仅捕获程序中的同步操作，内存操作的检查在硬件性能监控单元进行，进一步降低了开销。为了降低采样技术的运行开销，Cai 等[19]提出了一种新的数据竞争定义——时钟竞争，以满足低开销采样的需求，在检测中不依赖具体的锁，不直接延迟程序执行，并且避免了发生序关系跟踪时的基本开销。Guo 等[20]添加足够多的测试用例，实现了一个跨线程间和跨测试间的采样方法，缩短了运行时间。

单独使用发生序关系或者锁集法会暴露各自的缺点，许多研究人员将两种方法的优点相结合，以达到更好的检测效果。Li 等[21]结合 Happens-before 关系和锁集法，利用时钟向量计算 BPEL 基本块的 Happens-before 关系，检查是否并发、共享变量的读写访问情况以及是否有锁保护，进而判断多线程程序是否存在缺陷。Chen 等[22]将数据竞争的静态检测算法分为分析层、表示层和发现层，基于发生序和锁集关系建立数据竞争模型，使用拓扑排序获取数据竞争序列，提高了检测的准确度。

模型检测技术使用穷尽搜索进行自动化遍历或符号化不动点计算，通过验证系统模型是否满足规约和准则要求来判断是否存在缺陷。一些静态检测方法使用模型检测理论，提高了检测精度，减少了误报率。Nakade 等[23]模拟发生顺序，对竞争进行真实性筛选，得到缺陷代码的数据竞争故障。Taft 等[24]基于模型检测理论，分析程序中含有锁的执行路径并对结果进行二次过滤，提高了检测精度。Royuela 等[25]基于 OpenMP 和同步任务同时识别潜在的数据竞争。但模型检测技术存在搜索空间爆炸的问题，为此，学者们[26-28]对搜索空间进行压缩和剪枝，一定程度上缓解了此问题。

综上所述，本文对常用方法从人工干预度、扩展度、误报率、漏报率和执行开销 5 个方面进行了比较，如表 1 所列。

表 1 静态检测技术比较

Table 1 Comparison of static detection technology

方法技术	人工干预度	扩展性	误报率	漏报率	执行开销
数据流分析	低	一般	较高	较低	低
锁集法	低	一般	较高	较低	低
混合法	较低	一般	较高	较低	低
模型检测	高	差	一般	一般	低

3.2 动态分析技术

动态分析使用插桩技术获取共享变量和别名的相关信息，人工干预度较低，并且动态地运行程序，得到的都是真实存在的问题，因此误报率低。但由于在程序运行过程中，线程每次的执行顺序具有不确定性，导致每一次执行程序得到的结果不同[5]，因此动态检测不够全面，覆盖面不广，导致漏报率较高，并且动态监控运行会造成较大的执行开销。本节总结了数据竞争检测主要的动态分析技术。

Djit+[29]是一种基于 Happens-before 关系的动态分析技术，记录共享内存单元进行历史读 VC 和历史写 VC 的操作，利用向量时钟判定读写操作的先后顺序，进而判断数据竞争。在此基础上，FastTrack[30]采用轻量级逻辑时钟代替向量锁机制，只记录每个共享内存最后一次写操作的信息，大大降低了时间复杂度，使得检测算法在时间上(接近于 O(1))得到很大的提升。但由于 Djit+系列的检测工具都是基于 Happens-before 思想，对线程调度顺序有一定的限制，故该方法检测不够全面，具有一定局限性。Savage 等[8]最早将锁集算法引入数据竞争动态分析中，并开发了工具 Eraser 来动态检测数据竞争。基于锁集合的动态检测技术提高了检测效率，但忽略了其他内存变量，因此误报率相对较高。

早期的动态检测技术存在运行开销大或者漏报率高等问题。为了提高检测精度，学者们从降低执行开销、减少假阳性和假阴性、数据竞争再分类 3 个方面进行了优化。

(1)降低执行开销。一些研究者对 FastTrack 进行优化，例如：SOS[31]结合了一种基于观察的优化技术，其平均开销仅为 FastTrack 的 45%；Li 等[32]在此基础上加入锁模式，监控同时运行的并发线程函数对，通过预竞争检测获取真正涉及数据竞争的内存访问对，降低分析开销；根据共享内存位置的不同，Song 等[33]基于矢量时钟对检测粒度进行动态调整，降低了内存消耗；AdaptiveLock[34]使用更简单的数据结构和消除冗余操作，基于多线程程序中多个锁很少引发数据竞争这一理论，用一个简单的比较操作代替锁集合，在不引入额外假阳性和假阴性的同时提高检测速度；同样，HisLock+[35]减少了锁集比较，检测速度明显提高；Valor[36]通过减少以往方法对每一个读写操作的全部分析，只跟踪共享变量最后读取区域，从而得到高性能；PARSNIP[37]和 SLIMFAST[38]采用新的元数据编码，通过元数据无损压缩大幅减少空间的使用，加快了检测速度；Kard[39]运用 MPK(内存保护键)保证共享变量只访问关键部分的单个线程，并自动删除不活跃或冗长的缺陷，检测不一致锁使用引起的并发问题，降低运行开销，但删除部分信息导致漏报较高；为了暴露和检测隐藏的数据竞争，Yu 等[40]使用噪声注入干扰线程之间的锁定序列，找出隐藏的数据竞争，具有较低的运行时开销。

(2)减少假阳性和假阴性能够有效提高检测精确度。ACCULOCK[41]使用事件子集推理分析程序的执行，并应用一种新的锁集算法，区分读和写验证锁规则，减少了误报。Eslamimehr 等[42]优化了 Chord 工具的执行搜索方法，最终找到了正确的输入和调度，但有一定的局限性。Huang 等[43]将数据竞争检测问题转化成约束求解问题，降低了误报率。针对动态分析技术漏报问题，Sun 等[44]在 hybrid 数据竞争指令级算法中进行分类预测，消除了漏报，降低了误报。DILP[45]在程序运行时监视执行交错，收集变量访问和执行函数信息，检测不一致锁保护导致的数据竞争，找到了一些未发现的真实缺陷，保证了精度。

(3)一些检测到的数据竞争并不是有害竞争，对其再分类能够提高检测精度。一些方法[46-53]对初步检测到的可疑数据竞争进行分类，得到有害数据竞争和良性数据竞争。Narayanasamy 等[46]在重放过程中对数据竞争进行两次重复执行，通过优先级排序自动将数据竞争分为有害和良性两类，并向程序员提供缺陷的可复制场景。Frost[47]通过执行带有互补线程的程序的多个副本，保护程序不受误判数据竞争的干扰。Pordend[48]将多路径和多线程调度组合起来，分析比较潜在序列并自动分类，提高了准确度。RaceChecker[49]结合同步和 Happens-Before 关系修剪不可行的竞争，将潜在竞争分组，并同时对每组进行验证，提高了检测效率。Zhu 等[50]利用 Hadoop 分布式系统对可疑数据竞争进行并行检测，提出分层过滤方法，以减少原始踪迹中的冗余数据，在多个云服务器中控制线程调度，同时验证有害数据竞争，有效改善了检测精度。

表 2 对动态检测技术方法进行了总结，列出了动态分析优化算法的原理和特点。

表 2 动态检测技术方法

Table 2 Dynamic detection technology and method

优化方法	原理介绍	解决的问题
传统方法	使用传统 happens-before 关系和锁集法，运行开销大，漏报率高	—
缩小检测区域	通过优先排序、噪声注入、只访问关键部分等方法，减少不必要的分析和比较	降低了执行开销
执行控制技术	在程序运行时控制线程调度，优化搜索方法	提高了检测准确度
竞争再分类	对可疑序列进行分类，并分组进行二次验证，得到有害序列	提高了检测准确度

3.3 混合检测技术

除以上两种主要的数据竞争检测技术外，研究人员还提出了一些其他的方法：一些学者将静态分析方法和动态分析方法相结合；还有一些研究者将这两种方法中的一种与其他技术相结合；此外，还有纯硬件检测方法。

将动态分析与静态分析相结合能够利用其各自的优点，大幅降低误报率和漏报率，提高检测准确率。一般方法为：根据静态分析漏报率低的特点，先检测出所有可疑数据竞争，然后使用动态分析技术验证是否是真正的数据竞争。Kasicki 等[54]提出的 RaceMob 首先使用锁集法静态分析出所有可疑竞争，然后动态监测与可疑竞争相关的内存访问，保证了较低的

运行开销和较高的准确性;并且 Kasicki 等提出 Portend+[55]对数据竞争进行分类,根据数据竞争的严重等级分析潜在的后果。Yang 等[56]对 RaceChecker 进行改进,使用动态插桩对第一步的结果进行验证,一定范围内减少了静态分析检测结果的误报。SmartTrack[57]对以往检测算法进行了优化,加快了检测速度。Ding 等提出的 RaceTest[52]技术将重点放在有害数据竞争上,首先通过静态分析获取潜在的竞争对,生成包含可行潜在竞争的测试用例,对其进行动态分析,识别有害的数据竞争,有效减少了误报。

一些研究者将传统检测方法与其他技术相结合,减少了结果的假阳性和假阴性,一定程度上降低了执行开销。目前的研究一般与数据挖掘、模糊测试和切片技术相结合。

从基于数据挖掘技术的源代码中推断编程规则已被证明是检测并发错误的有效方法。MUVI[58]是第一个检测多变量访问不一致的工具,结合静态程序分析和数据挖掘技术,通过挖掘变量配对规则来检测更新错误和并发错误。数据挖掘分类模型可以高效处理大量数据,Sun 等根据这一特点实现了基于随机森林的数据竞争指令级检测工具 AIRaceTest[44]。Befrouei 等[59]使用一种序列模式挖掘的数据挖掘技术,该技术识别多线程共享内存中的问题读写序列,并通过减少执行跟踪长度来找出问题序列,加快了检测速度。NAR-Miner[60]从大型系统中自动提取负关联规划规则,将规则挖掘集中在语义关系较强的元素上,有效地缓解了爆炸问题。

模糊测试技术是一种有效、实用的检测并发缺陷的方法,通常与传统检测技术相结合,通过在检测时提供非预期的输入来监视异常状况,发现缺陷。Razzer[61]先通过传统的静态分析定位所有可疑竞争对,然后运用模糊技术进行确认,在准确度和检测效率上都有大幅提高。ConFuzz[62]将传统方法与模糊化技术相结合,最大限度地提高了在程序中查找新调度和路径的可能性。在动态检测过程中,Krace[63]通过 3 个新的构造将覆盖率引导的模糊化引入其中,捕捉程序的探索过程、Happens-before 关系以及锁集信息,并对多线程序列进行生成、变异与合并,扩大了工具的检测范围,减少了运行时间。

通过有效缩小或合并路径,切片技术能够减少程序的路径空间,提高检测方法的可扩展性。Zheng 等[64]将动态切片技术集成到约束求解中,简化了程序内存访问。Zhang 等[65]通过控制流分析和切片技术对 Happens-before 关系进行判断,并加入别名分析来提高精度,减少发生序方法中出现的误报问题。Ghosh 等[66]在缺陷检测时加入基于频谱的动态切片技术,根据切片准则将无关代码剔除,缩短缺陷定位时间,只需检查部分代码就能检测出数据竞争。

许多检测工具使用符号执行技术来修剪并发程序中的冗余执行,缓解程序执行时的空间压力,节省程序遍历时间,但该技术识别可能导致错误的可疑程序位置,并沿着这些位置赋予变量符号化,导致生成错误状态的执行路径。Yu 等[67]通过符号正则验证来修剪多余的路径,大大减少了探索路径的数量。Wang 等[68]提出了一种新的符号执行方法来检测故障,该方法利用程序依赖关系,基于符号值冗余线路的预测和消除,不需要对路径进行系统探索,缓解了传统方法带来的路径爆炸问题,表现出了良好的故障检测能力。

一些工具使用纯硬件的方法检测数据竞争,纯硬件方法在一定程度上可以提高检测覆盖率和减少性能开销。NUDA[69]和 RaceSMM[70]通过添加新的硬件构架来连接所有处理内核,RaceSMM 在缓存架构中使用元数据,以便通过硬件更快地检测数据竞争,但由于数据开销可能很大以及元数据的处理时间可能较长,故该工具仅针对最近共享的内存位置有选择地存储元数据。NUDA 构建了一个环形互联网络,以收集所需的访问模式,降低了检测成本。Song 等[71]在硬件平台上进行改进,将数据竞争检测与应用线程分开,在检测线程中执行分析,每个检测线程只分析单一区域内存访问,在平均运行速度上有了大幅提升。Sahneh 等[72]提出了一种线程映射算法,最大限度地减少了多线程程序线程之间的缓存竞争条件,通过在内核上分配现有的和新的线程,将缓存争议降到最低,解决了性能成本相对较高的问题。但纯硬件方法一般需要修改体系框架和重新设计缓存协议,通用性较差。

4 目前存在的问题

4.1 部分工具对比分析

本文选取了上述数据竞争检测技术中的 21 个代表性研究工作进行分析比较,如表 3 所列。第 1 列表示该项研究工作的检测技术类别,分别为静态分析、动态分析和混合方法;第 2 列为数据竞争检测工具名称或研究人员姓名;第 3 列为每个工具的基准测试用例;第 4 列为该工具检测到的数据竞争数目及漏报、误报情况;第 5 列为该工具进行数据竞争检测时的检测速度及执行开销。

(1)准确度是衡量缺陷检测工具好坏的最基本因素。通过调查发现,大多数开发人员会采纳检测工具给出的结果,因此一个高效准确的检测工具非常重要。理想情况下,检测工具应该检测到每一个数据竞争,并且不对正确代码发出错误警告。在表 3 中,大多数工具都有漏报或误报的情况发生。产生误报的原因主要有以下两方面:1)未能获得足够的程序信息而导致的虚假错误;2)一些数据竞争是良性数据竞争,不影响程序的正确性,这些竞争通常是开发人员出于性能方面的考虑设计的,产生漏报的主要原因是一些工具在检测时未能获取全部信息。

(2)执行开销和检测速度是评估工具的另一重要因素。动态检测工具在每次运行时会对共享变量的加载和存储、加锁和解锁调用、以及初始化和内存分配等进行检测,这些操作在一定程度上会影响性能。从表 3 可以看出,一些研究人员只对内存访问的核心部分进行检测,以准确率换取运行开销,或者舍弃一些不必要的信息,分配多条线路并行检测。总体来看,表中静态分析的执行开销比动态分析及混合方法低,但如果程序的输入、路径和状态有无限多个,执行开销会相对增加。混合方法中,研究人员为了进行更精确的分析,开销都会有少量增加。纯硬件方法能够在一定程度上降低执行开销,加快检测速度。

表 3 数据竞争检测技术对比分析

Table 3 Comparison of detection technology for data race

类别	工具名称	测试程序	检测结果	检测速度/执行开销
静态分析	RacerX[10]	Linux kernel v2.5.62, System X	System X:38 个数据竞争,其中 7 个已确认,14 个误报 Linux kernel v2.5.62:13 个数据竞争,其中 3 个已确认,6 个误报	2-14min/1.8MLOC
	Chord[11]	tsp,hedc,ftp 等	Tsp:19 个数据竞争,其中 7 个确认为真正数据竞争,12 个误报 Hedc:17 个数据竞争,其中 4 个确认为真正数据竞争,13 个误报 ftp:71 个数据竞争,其中 45 个确认为真正数据竞争,23 个误报	Tsp:1 m3 s/76KLOC Hedc:1 m10 s/83KLOC ftp:1 m17 s/103.2KLOC
	Conracer[13]	IBM Contest,JGF,Dacapo 等基准套件	42 个数据竞争,其中 1 个为误报	93.3s/525.9KLOC
	Relay[14]	Linux v2.6.15	149 个数据竞争,其中 53 个确认为真正数据竞争	未并行化:72 h/4.5MLOC 优化后:5 h/4.5MLOC
	Locksmith[15]	POSIX Apps, Linux Drivers	POSIX Apps:发出 93 个警告,其中报告 26 个数据竞争 Linux Drivers:发出 91 个警告,其中报告 4 个数据竞争	11.1 s/14.4KLOC 7.9 min/315.4KLOC
动态分析	FastTrack[30]	colt,tsp 等 16 个实验基准	检测到 8 个数据竞争	比 Djit+快 2.3 倍,分配超过 7.9 亿个矢量时钟
	AdaptiveLock[34]	Java 上 17 个基准程序,共两组	第一组小规模基准程序:数据竞争被全部检出 第二组大规模基准程序:检测到 26 个数据竞争,有 4 个漏报	内存消耗与 FastTrack 基本持平,但运行时间比 FastTrack 长
	Kard[39]	PARSEC,SPLASH-2x 基准测试套件和 4 个真实程序	真实程序中共检测出 6 个数据竞争,其中 1 个为误报	与基线相比性能开销平均值仅为 7.0%
	ACCULOCK[41]	11 个基准程序:6 个 DaCapo9.12 bach 基准和 5 个真实程序,共 1.1MLOC	共报告 257 个数据竞争警告,比 FastTrack 多 72 个警告,其中 3 个为误报	与基本运行时间相比平均增加 10.9%,平均执行开销 4.47/66 MB
	Huang 等[43]	IBM Contest 基准套件、Java Grande 基准和真实应用程序,共 1.7MLOC	共检测出 299 个数据竞争,其中 11 个为之前未发现的竞争并被开发人员确认	0.4 s~30 min
	AIRaceTest[44]	Parsec3.1 基准程序	共检测出 170 个数据竞争,误报率为 10.3%,漏报率为 6.6%	相比 Eraser 和 Djit+,时间开销平均降低 41.8%,内存开销平均降低 22.4%
	DILP[45]	Linux kernel v3.3.1, v4.16.9	Linux kernel v3.3.1 中发现 13 个数据竞争,Linux kernel v4.16.9 中发现 25 个数据竞争。最终 11 个被确认为真正数据竞争	平均 7.2x 的开销
	Satish 等[46]	18 个 Windows Vista 和 Internet Explorer 中重放日志	共检测到 68 个数据竞争,包括 32 个良性(47%)和 36 个有害(53%),其中只有 7 个被验证为有害数据竞争(20%)	—
	Zhu 等[50]	Atom-aid,SPLASH2	共检测到 29 个有害数据竞争,有 3 个漏报	与 RaceChecker 相比,运行时间平均减少 46%
	ConVul[53]	10 个已知的并发漏洞和 MySQL 数据库服务器	10 个已知的并发漏洞:检测到 9 个,1 个漏报 MySQL 数据库服务器:检测到 6 个数据竞争,其中 4 个已被确认	—
混合方法	RaceTest[52]	11 个开源 WS-BPEL 程序	共检测到 36 个有害数据竞争,并在 11 个 WS-BPEL 中过滤 16 个良性数据竞争,有 1 个漏报	运行时间长于 RaceMob
	MUVI[58]	Linux Drivers, Mozilla, MySQL, PostgreSQL	检测到 39 个新的错误,其中 17 个被开发人员确认在 Mozilla 中检测出 9 个新的错误	共运行 450m30s/9.7MLOC 增加 14%的执行开销
	Razzer[61]	Linux kernel v4.16-rc3, v4.18-rc3	检测出 30 个数据竞争问题,其中 16 个已被开发人员确认	7 day
	KRACE[63]	Linux kernel v5.4-rc5 文件系统 ext4, btrfs 和 VFS 中	检测出 23 个数据竞争,其中 9 个数据竞争已被确认为有害数据竞争	模糊测试:7 day 缺陷检测:96 h
	SRD[65]	IBM Contest,JGF,Dacapo 等基准套件	检测出 52 个数据竞争,其中 14 个为误报	36s/525.8KLOC
	RaceSMM[70]	Mozilla,MySQL, Apache,SPLASH2	检测出 447 个数据竞争,有 3 个漏报	平均性能开销 4.8% 最坏性能开销 12.5%

4.2 总体评估

通过对相关工具的分析统计,我们对文献中的所有检测技术从检测精度、执行开销和扩展性 3 个方面进行总体评估,如表 4 所列。

表 4 检测技术总体评估

Table 4 Overall evaluation of detection technologies

检测技术	误报率	漏报率	执行开销	扩展性
静态分析	较高	较低	低	较差
动态分析	较低	较高	高	较好
动静结合	较低	较低	较高	较好
传统+其他技术	较低	较低	较低	一般
硬件方法	较高	较高	低	较差

从表 4 可以看出:

(1)静态分析能够有效覆盖所有执行路径,不会有漏报,但由于缺少程序交替执行时的一些信息,会有大量误报。动态分析虽然误报率较低,但其依赖线程调度信息,导致一些没有运行或者隐藏的数据竞争不会被发现,会产生较多的漏报。并且动态分析需要输入,若测试人员没有参与程序的前期开发,可能会没有足够的相关领域知识来进行有效输入,导致动态执行时覆盖率不高,检测深度不够。混合方法改善了单一检测的缺点,相比其他两种方法误报率和漏报率都有明显降下。硬件方法主要是对执行开销和检测速度进行改进,在

准确率上没有明显的效果。

(2)在动态检测时,需要对程序进行实际执行,通过增加线程调度暴露出更多的数据竞争,因此不管在时间还是空间上,运行开销都较大。静态检测不需要执行程序,因此检测速度和执行开销都较低,但要注意路径爆炸问题。动静结合的方法在对可疑竞争进行验证时,一般需要人工调试线程之间的调度,若可疑竞争数量较多,在实际的运行过程中也会产生较大开销。传统方法与其他技术相结合以及纯硬件方法都降低了一定的执行开销。

(3)一些程序比较庞大和复杂,有上百万、上千万行代码,线程和锁的数量也较多,在检测时面临任务繁琐和准确率不高的问题,必然会对检测工具的可扩展性提出更高的要求。一个好的数据竞争检测工具不仅要保持较高的检测精度,而且工具的性能应随着线程和锁数量的增加缓慢降低,而不是突然下降。目前的检测工具中,静态检测工具可扩展性较差,通常权衡精度和效率,很难完全分析一个大型程序;动态检测工具扩展性良好,但随着代码的增加会出现性能开销大的问题。

5 研究展望

在过去的十几年里,针对 CPU 并行程序的数据竞争检测技术已经逐渐走向成熟,学者们对其进行了大量研究,在缺陷检测、错误避免和代码修复方面都取得了很多实质性的成果。总体来看,现阶段大部分研究还是采用静态分析技术与动态分析技术相结合或者传统检测方法与其他技术相结合的方式,虽然取得了一定进展,但该领域还存在大量值得研究的问题。针对现有检测技术的问题,我们指出了未来数据竞争检测技术的一些关注问题和研究方向。

(1)更精确、有效的检测技术。当前的检测技术假阳性和假阴性偏高,并且缺陷报告展现不直观,不能给予程序员简洁明了的指导;没有从使用者的角度来判定这种结果是否符合预期,是否是程序设计者刻意为之。一些检测工具对数据竞争进行再分类时,有害和良性数据竞争界限模糊。未来应该开发准确率更高、更符合开发者实际设计预期的检测技术。

(2)检测及后续修复和规避的自动化实现。状态空间经过线程交错会变得更加复杂和庞大,因此非常需要检测、修复协同处理并发错误。现有的研究多为单独检测或者单独修复、规避,同时,一些修复工具会根据检测报告进行修复,若检测结果不准确,会导致修复出现错误或错过修复;并且即使检测结果准确,在修复过程中也可能产生其他错误而不能及时发现,造成不必要的麻烦。因此,若 3 个环节都实现自动化处理,将大大减少错误的发生。

(3)软硬件协同提高各方面性能。纯软件检测方法适用于不同的平台,适用性较强,但随着代码量的增加可能会对程序性能产生负面影响,降低执行速度。纯硬件方法可以加快执行速度,降低执行开销,但是可用性和通用性较差。未来的研究可以将软硬件相结合,充分利用两者的优点来检测数据竞争。

(4)减少运行开销、优化检测时间。在对一些大型程序进行并发缺陷检测时,会消耗大量时间,并且可能会导致空间和时间复杂度呈指数型增长,继而引发路径爆炸等问题。基于此,在开发检测技术时应该考虑剪枝、路径分配、启发式搜索、遍历优化、与其他技术相结合等方法,缓解空间遍历时的路径爆炸问题,提高检测效率。

(5)更有效的记录和重放支持。多线程程序运行时的不确定性导致线程交织执行复杂,动态分析技术对这一现象较为敏感,如果在本次执行交错中检测到某些数据竞争,那么下次当程序有新的输入或者新的执行路线时几乎不可能再次检测到该并发缺陷,将降低检测的有效性。重放机制能使某次执行具有可重复性,记录每次执行的情况,并进行重放,从而使数据竞争在新的执行中仍然能被检测出来。

(6)更符合用户需求的错误输出形式。现有的检测工具一般直接罗列出结果,没有进行详细的分类,例如仅简单告知一个共享变量的操作会产生数据竞争,提醒使用者修复错误。但在实际应用中,开发者为了提升性能或实现相互通信等而有意设计一些数据竞争,用户需要查看源代码中数据竞争所在位置的具体上下文信息来判断是否是真正的错误;而且一些错误发生的背景非常复杂,没有充足的信息很难快速作出判断,因此一个符合用户需求的输出形式是必不可少的。

(7)更有针对性的检测技术。现有的检测技术对主流开发平台的并发程序有较好的检测结果,但对特定平台的适用性有待提高,针对安卓、web、嵌入式等平台的数据竞争检测技术的研究比较匮乏。例如在嵌入式系统中,其在调度、同步机制等方面与其他开发平台有较大差异,同时,并发原语复杂多样,若使用现阶段的检测方法会产生大量误报。因此,如何将现有技术与特定平台相结合,实现高准确率的专用检测工具,也是未来需要关注的一个重要方向。

结束语 数据竞争检测技术在代码安全、程序分析等领域发挥着重要作用。近年来,针对 CPU 并行程序的数据竞争检测技术取得了显著的成果。本文在对大量相关文献进行调研后,总结了当前数据竞争检测相关工作的研究概况。首先介绍了数据竞争的基本概念、产生原因和主要检测思想;然后根据检测手段的不同,将现有检测方法分为静态、动态和混合法三大类,并从检测精度、执行开销、扩展性和可用性 4 个方面对这些技术进行分析和比较。目前该技术还存在一些挑战,如通用性低、开销大、准确率低、形式单一等,但是随着检测技术的发展,这些挑战终将被克服。

参 考 文 献

[1] WANG C,YUAN X L,CUI Y,et al. Toward Secure Outsourced Middlebox Services:Practices,Challenges,and Beyond[J]. IEEE Network,2017,32(1):166-171.

[2] LU S,PARK S,SEO E,et al. Learning from Mistakes—A Comprehensive Study on Real World Concurrency Bug Characteristics[J]. Computer Architecture News,2008,36(1):329-339.

[3] JU Y,HUANG Y H,ZHENG J T,et al. Multi-thread Parallel Algorithm for Reconstructing 3D Large-Scale Porous Structures [J]. Computers & Geosciences,2017,101:10-20.

[4] SEREBRYANY K, ISKHODZHANOV T. ThreadSanitizer-Data Race Detection in Practice[C]// Proceedings of the Workshop on Binary Instrumentation and Applications. New York: ACM, 2009: 62-71.

[5] SU X H, YU Z, WANGT T, et al. A Survey on Exposing, Detecting and Avoiding Concurrency Bugs[J]. Chinese Journal of Computers, 2015, 38(11): 2215-2233.

[6] DENG D D, ZHANG W, LU S. Efficient Concurrency-Bug Detection Across Inputs[J]. ACM Sigplan Notices, 2013, 48(1): 785-802.

[7] LAMPORT L. Time, Clocks, and the Ordering of Events in a Distributed System[J]. Communications of the ACM, 1978, 21(7): 558-565.

[8] SAVAGE S, BURROWS M, NELSON G, et al. Eraser: A Dynamic Data Race Detector for Multi-Threaded Programs[J]. ACM Transactions on Computer Systems, 1997, 15(4): 391-411.

[9] BO L L, JIANG S J, ZHANG Y M, et al. Research Progress on Techniques for Concurrency Bug Detection[J]. Computers Science, 2019, 46(5): 20-27.

[10] ENGLER D, ASHCRAFT K. RacerX: Effective, Static Detection of Race Conditions and Deadlocks[C]// Proceedings of the 19th ACM Symposium on Operating Systems Principles. New York: ACM, 2003: 237-252.

[11] NAIK M, AIKEN A, WHALEY J. Effective Static Race Detection for Java[J]. ACM Sigplan Notices, 2006, 41(6): 308-319.

[12] WU X G, CHEN L Q. Static Analysis of Runtime Errors in Interrupt-Driven Programs via Sequentialization[J]. ACM Transactions on Embedded Computing Systems, 2016, 15(4): 70-95.

[13] ZHANG Y, LIU H, QIAO L. Context-Sensitive Data Race Detection for Concurrent Programs[J]. IEEE Access, 2021, 9: 20861-20867.

[14] VOUNG J W, JHALA R, LERNER S. RELAY: Static Race Detection on Millions of Lines of Code[C]// Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on the Foundations of Software Engineering. Dubrovnik: ACM, 2007: 205-214.

[15] PRATIKAKIS P, FOSTER J S, HICKS M. LOCKSMITH: Context-Sensitive Correlation Analysis for Race Detection[J]. ACM SIGPLAN Notices, 2006, 41(6): 320-331.

[16] ANDRIANOV P, MUTILIN V, KHOROSHILOV A. Predicate Abstraction Based Configurable Method for Data Race Detection in Linux Kernel[C]// Proceedings of the International Conference on Tools and Methods for Program Analysis. Moscow: Springer, 2018: 11-23.

[17] CHOPRA N, PAI R, DSOUZA D. Data Races and Static Analysis for Interrupt-Driven Kernels[C]// Proceedings of 28th European Symposium on Programming (ESOP). Prague: Springer, 2019: 697-723.

[18] SHENG T W, VACHHARAJANI N, ERANIAN S, et al. RACEZ: A Lightweight and Non-Invasive Race Detection Tool for Production Applications[C]// Proceedings of the IEEE 33th International Conference on Software Engineering. Hawaii: IEEE, 2011: 401-410.

[19] CAI Y, ZHANG J, CAO L W, et al. A Deployable Sampling Strategy for Data Race Detection[C]// Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press, 2016: 810-821.

[20] GUO Y, CAI Y, YANG Z J. AtexRace: Across Thread and Execution Sampling for In-House Race Detection[C]// Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2017: 315-325.

[21] LI S D, XU L. A Static Data Race Detecting Method for BPEL Based on Happens-Before and Lockset[J]. Computer & Digital Engineering, 2010, 38(8): 6-9.

[22] CHEN J, ZHOU K J, JIA M. Multi-thread parallel program data race static detection model[J]. Computer Engineering and Design, 2017, 38(5): 1264-1272.

[23] NAKADE R, MERCER E, ALDOUS P, et al. Model-Checking Task-Parallel Programs for Data-Race [J]. Innovations in Systems and Software Engineering, 2019, 15(1): 367-382.

[24] TAFT S T, SCHANDA F, MOY Y. High-Integrity Multitasking in SPARK: Static Detection of Data Races and Locking Cycles[C]// Proceedings of the IEEE 17th International Symposium on High Assurance Systems Engineering. Orlando: IEEE, 2016: 238-239.

[25] ROYUELA S, MARTORELL X, QUIÑONES E, et al. Safe Parallelism: Compiler Analysis Techniques for Ada and OpenMP [C]// Proceedings of the 23rd International Conference on Reliable Software Technologies. Portugal: Springer, 2018: 141-157.

[26] KAHLON V, SANKARANARAYANAN S, GUPTA A. Static Analysis for Concurrent Programs with Applications to Data Race Detection[J]. International Journal on Software Tools for Technology Transfer, 2013, 15(4): 321-336.

[27] HUANG J. Stateless Model Checking Concurrent Programs with Maximal Causality Reduction[J]. ACM SIGPLAN Notices, 2015, 50(6): 165-174.

[28] BLUM B, GIBSON G. Stateless Model Checking with Data-Race Preemption Points[J]. ACM SIGPLAN Notices, 2016, 51(10): 477-493.

[29] POZNIANSKY E, SCHUSTER A. Efficient On-the-Fly Data Race Detection in Multithreaded C++ Programs[J]. Concurrency and Computation: Practice & Experience, 2003, 19(3): 327-340.

[30] FLANAGAN C, FREUND S N. FastTrack: Efficient and Precise Dynamic Race Detection[J]. Communications of the ACM, 2010, 53(11): 93-101.

[31] LI D, SRISA W, DWYER M B. SOS: Saving Time in Dynamic Race Detection with Stationary Analysis[J]. ACM SIGPLAN Notices, 2011, 46(10): 35-50.

[32] LI M K, ZHENG Q S, WANG L. Dynamic Hybrid Data Race Detection Algorithm Based on Sampling Technique[J]. Computers Science, 2020, 47(10): 315-321.

[33] SONG Y W, LEE Y H. Efficient Data Race Detection for C/C++ Programs Using Dynamic Granularity[C]// Proceedings of the 2014 IEEE International Parallel & Distributed Processing Symposium. Arizona: IEEE, 2014: 679-688.

[34] YU M,LEE J,BAE D,et al. AdaptiveLock:Efficient Hybrid Data Race Detection Based on Real-World Locking Patterns[J]. International Journal of Parallel Programming,2019,47(7):805-837.

[35] YANG J,JIANG B,CHAN W K. HistLock+:Precise Memory Access Maintenance Without Lockset Comparison for Complete Hybrid Data Race Detection[J]. IEEE Transactions on Reliability,2018,67(3):786-801.

[36] BISWAS S,ZHANG M J,BOND M D,et al. Valor:Efficient,Software-Only Region Conflict Exceptions[C]// Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming,Systems,Languages,and Applications. New York:ACM,2015:241-259.

[37] PENG Y F,WOOD B P,DEVIETTI J. PARSNIP:Performant Architecture for Race Safety with No Impact on Precision[C]// Proceedings of the 50th Annual IEEE/ACM International Symposium. New York:ACM,2017:490-502.

[38] PENG Y F,DELOZIER C,EIZENBERG A,et al. SLIMFAST:Reducing Metadata Redundancy in Sound and Complete Dynamic Data Race Detection[C]// Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium(IPDPS). New York:IEEE,2018:835-844.

[39] AHMAD A,LEE S,FONSECA P,et al. Kard:Lightweight Data Race Detection with Per-Thread Memory Protection[C]// Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. NewYork:ACM,2021:647-660.

[40] YU M,MA Y S,BAE D H. Correction to:Efficient Noise Injection for Exposing Hidden Data Races[J]. The Journal of Supercomputing,2020,76(21):1-32.

[41] XIE X W,XUE J L,ZHANG J. Acculock:Accurate and Efficient Detection of Data Races[J]. Software Practice & Experience,2013,43(5):543-576.

[42] ESLAMIMEHR M,PALSBERG J. Race Directed Scheduling of Concurrent Programs[J]. ACM SIGPLAN Notices,2014,49(8):301-314.

[43] HUANG J,MEREDITH P O,ROSU G. Maximal Sound Predictive Race Detection with Control Flow Abstraction[C]// Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York:ACM,2014:337-348.

[44] SUN J Z,YANG J W,YANG Z J. Random forest instruction level detection model for data race in multithreaded programs[J]. Journal of Tsinghua University(Science and Technology),2020,60(10):804-813.

[45] CHEN Q L,BAI J J,JIANG Z M,et al. Detecting Data Races Caused by Inconsistent Lock Protection in Device Drivers[C]// Proceedings of the 2019 IEEE 26th International Conference on Software Analysis,Evolution and Reengineering(SANER). Hangzhou:IEEE,2019:366-376.

[46] NARAYANASAMY S,WANG Z H,TIGANI J,et al. Automatically Classifying Benign and Harmful Data Races Using Replay Analysis[J]. ACM SIGPLAN Notices,2007,42(6):22-31.

[47] VEERARAGHAVAN K,CHEN P M,FLINN J,et al. Detecting and Surviving Data Races using Complementary Schedules[C]// Proceedings of the 23th ACM Symposium on Operating Systems Principles. New York:ACM,2011:369-384.

[48] KASIKCI B,ZAMFIR C,CANDEA G. Data Races vs. Data Race Bugs:Telling the Difference with Portend[J]. ACM SIGPLAN Notices,2012,47(4):185-198.

[49] KAI L,WU Z,WANG X,et al. RaceChecker:Efficient Identification of Harmful Data Races[C]// Proceedings of the 2015 23rd Euromicro International Conference on Parallel,Distributed and Network-Based Processing(PDP). Turku:IEEE,2015:78-85.

[50] ZHU S X,WU Y N,SUN G L,et al. A Concurrent Harmful Races Identification Algorithm using Hadoop and Multiple Cloud Servers[J]. International Journal of Performability Engineering,2018,14(10):2332-2342.

[51] MAJEED S,RYU M. Debugging Nondeterministic Failures in Linux Programs through Replay Analysis[J]. Scientific Programming,2018(PT.1):1-11.

[52] DING Z J,ZHOU Z X. RaceTest:harmful data race detection based on testing technology in WS-BPEL[J]. Service Oriented Computing and Applications,2019,13(5):141-154.

[53] CAI Y,ZHU B Y,MENG R J,et al. Detecting Concurrency Memory Corruption Vulnerabilities[C]// Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. NewYork:ACM,2019:706-717.

[54] KASIKCI B,ZAMFIR C,CANDEA G. RaceMob:Crowdsourced Data Race Detection[C]// Proceedings of the 24th ACM Symposium on Operating Systems Principles. Farmington:ACM,2013:406-422.

[55] KASIKCI B,ZAMFIR C,CANDEA G. Automated Classification of Data Races Under Both Strong and Weak Memory Models[J]. ACM Transactions on Programming Languages and Systems,2015,37(3):1-44.

[56] YANG Z,YU Z,SU X H,et al. RaceTracker:Effective and Efficient Detection of Data Races[C]// Proceedings of the 17th IEEE/ACIS International Conference on Software Engineering. Shanghai:IEEE Computer Society,2016:293-300.

[57] ROEMER J,GEN K,BOND M D. SmartTrack:Efficient Predictive Race Detection[C]// Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation(PLDI). London:ACM,2020:747-762.

[58] LU S,PARK S,HU C,et al. MUVI:Automatically Inferring Multi-Variable Access Correlations and Detecting Related Semantic and Concurrency Bugs[J]. ACM SIGOPS Operating Systems Review,2007,41(6):103-116.

[59] BEFROUEI M T,WANG C,WEISSENBACHER G. Abstraction and Mining of Traces to Explain Concurrency Bugs[J]. Formal Methods in System Design,2016,49(1/2):1-32.

[60] BIAN P,LIANG B,SHI W C,et al. NAR-miner:Discovering Negative Association Rules from Code for Bug Detection[C]// Proceedings of the 2018 26th ACM Joint Meeting on European

Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2018: 411-422.

[61] JEONG D R, KIM K, SHIVAKUMAR B, et al. Razzer: Finding Kernel Race Bugs through Fuzzing[C]// Proceedings of the 2019 IEEE Symposium on Security and Privacy(SP). San Francisco: IEEE, 2019: 754-768.

[62] PADHIYAR S, SIVARAMAKRISHNAN K C. ConFuzz: Coverage-Guided Property Fuzzing for Event-Driven Programs[C]// Proceedings of the 23th International Symposium on Practical Aspects of Declarative Languages(PADL). Copenhagen: ACM, 2021: 127-144.

[63] XU M, KASHYAP S, ZHAO H Q, et al. Krace: Data Race Fuzzing for Kernel File Systems[C]// Proceedings of the 2020 IEEE Symposium on Security and Privacy(SP). San Francisco: IEEE, 2020: 1643-1660.

[64] ZHENG L, LIAO X F, JIN H, et al. Towards Concurrency Race Debugging: An Integrated Approach for Constraint Solving and Dynamic Slicing[C]// Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques. Limassol: ACM, 2018: 1-13.

[65] ZHANG Y, LIANG Y N, ZHANG D W. SRD: Static Data Race Detection for Concurrent Programs[J]. International Journal of Performability Engineering, 2018, 11(14): 2683-2691.

[66] GHOSH D, SINGH J. A Dynamic Slicing-Based Approach for Effective SBFL Technique[J]. International Journal of Computational Science and Engineering, 2021, 24(1): 98-107.

[67] YU H B, CHEN Z B, WANG J, et al. Symbolic Verification of Regular Properties[C]// Proceedings of the 40th International Conference on Software Engineering. New York: ACM, 2018: 871-881.

[68] WANG H J, LIU T, GUAN X H, et al. Dependence Guided Symbolic Execution[J]. IEEE Transactions on Software Engineering, 2017, 43(3): 252-271.

[69] WEN C N, CHOU S H, CHEN T F, et al. NUDA: A Non-Uniform Debugging Architecture and Nonintrusive Race Detection for Many-Core Systems[J]. IEEE Transactions on Computers, 2012, 61(2): 199-212.

[70] HUANG R R, HALBERG E, FERRAIUOLO A, et al. Low-Overhead and High Coverage Run-Time Race Detection through Selective Meta-Data Management[C]// Proceeding of the 20th International Symposium on High Performance Computer Architecture(HPCA). Orlando: IEEE, 2014: 96-107.

[71] SONG Y W, LEE Y H. A Parallel FastTrack Data Race Detector on Multi-core Systems[C]// Proceeding of the 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Orlando: IEEE, 2017: 387-396.

[72] SAHNEH P S, SARIHI A, WARBURTON B, et al. RaceR: A Thread Mapping Algorithm for Race Reduction in Multi-Level Shared Caches[C]// Proceeding of the 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing(PDP). Pavia: IEEE, 2019: 228-232.

ZHAO Jing-wen, born in 1995, Ph. D candidate. Her main research interests include concurrent program analysis and compi-ler technology.

FU Yan, born in 1978, master, lecturer, is a member of China Computer Federation. Her main research interests include computer architecture and compiler technology.

(责任编辑:柯颖)