并发缺陷检测技术研究进展

薄莉莉[1] 姜淑娟[1] 张艳梅[1] 王兴亚[2] 于 巧[3]
(中国矿业大学计算机科学与技术学院矿山数字化教育部工程研究中心 江苏 徐州 221116)[1]
(南京大学计算机软件新技术国家重点实验室 南京 210093)[2]
(江苏师范大学计算机科学与技术学院 江苏 徐州 221116)[3]

摘 要 多核时代的到来使得并发程序的设计备受人们关注。然而,并发程序的并发性和不确定性容易引发并发缺陷。因此,快速且有效地检测出这些并发缺陷尤为重要。首先,将目前常见的并发缺陷分为五大类(并发类型状态缺陷、死锁、数据竞争、原子性违背和顺序违背);随后,从软件运行的角度,将现有的并发缺陷检测技术分为静态分析、动态分析和动静结合分析,并对每一类进行详细的分析、比较和总结;接着,对并发缺陷检测技术的通用性进行分析和总结;最后,从通用准确的并发缺陷检测、软硬件相结合的并发缺陷检测、并发缺陷检测修复一体化、适用于松散内存模型的并发缺陷检测、安卓等其他应用平台的并发缺陷检测和分布式系统非确定性并发缺陷研究等方面,对并发缺陷检测技术的未来研究进行了探讨。

关键词 并发程序,并发缺陷,缺陷检测,软件测试
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.05.002

Research Progress on Techniques for Concurrency Bug Detection

BO Li-li[1] JIANG Shu-juan[1] ZHANG Yan-mei[1] WANG Xing-ya[2] YU Qiao[3]
(Mine Digitization Engineering Research Center of the Ministry of Education, School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangsu 221116, China)[1]
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)[2]
(School of Computer Science and Technology, Jiangsu Normal University, Xuzhou, Jiangsu 221116, China)[3]

Abstract The advent of multi-core era makes concurrent programming more and more popular. However, concurrent programs could easily lead to concurrency bugs due to their inherent concurrency nature and non-deterministic thread scheduling. It is critical to detect concurrency bugs effectively and efficiently. First, concurrency bugs were divided into five categories(i.e., type-state bug, deadlock, data race, atomicity violation and order violation). Then, concurrency bug detection techniques were classified into static analysis, dynamic analysis and the combination analysis in items of running programs, each with the detailed analysis, comparisons as well as accurate summarizations. Next, the universality of the existing detection techniques was analyzed. Finally, the research directions on concurrency bug detection in the future were discussed.

Keywords Concurrent program, Concurrency bug, Bug detection, Software testing

1 引言

在当前多核架构充分普及的硬并发时代,并发程序以资源利用率高、计算速度快等优点,在现代软件开发中得到了广泛应用[1]。然而,由于多线程调度的不确定性,并发程序容易产生各种类型的并发缺陷,并造成严重事故,如 Therac-25 医疗事件[2]和 2003 年北美大停电事故[3]。以上机遇和挑战,激发了国内外广大研究学者对并发软件测试的研究热潮。对并发软件测试的研究最早可追溯到 Taylor 提出并发状态(Concurrency State)、并发历史(Concurrency History)等相关概念及形式化定义,并提出基于并发历史的并发分析方法及相关的结构性测试覆盖准则[4-5]。随后,研究者对该领域展开深入的研究,并提出了许多行之有效的方法。

国内外已有一些对并发软件调试和测试的综述性论

到稿日期:2018-08-15 返修日期:2018-10-16 本文受国家自然科学基金(61673384,61502497),南京大学计算机软件新技术国家重点实验室创新项目(ZZKT2018B02),江苏师范大学博士学位教师科研支持项目(17XLR001)资助。
薄莉莉(1989-),女,博士生,CCF 会员,主要研究领域为并发程序分析与测试等,E-mail:lilibo@cumt.edu.cn;**姜淑娟**(1966-),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为编译技术、软件工程等,E-mail:shjjiang@cumt.edu.cn(通信作者);**张艳梅**(1982-),女,博士,副教授,硕士生导师,主要研究领域为软件分析与测试等;**王兴亚**(1990-),男,博士,主要研究领域为软件分析与测试等;**于 巧**(1989-),女,博士,主要研究领域为软件缺陷预测、机器学习等。

文[6-10]。为了对该研究问题进行全面的分析、比较和总结，我们对近 12 年来在并发软件测试领域顶级会议和期刊上的文章进行调研。首先在 IEEE，ACM，Springer，Elsevier 和 CNKI 等论文数据库中进行检索，然后对检索出的论文进行人工审查，以筛选出与该研究问题直接相关的论文。最终，选择出高质量论文共 199 篇（截止到 2017 年 12 月）。其中，绝大部分论文发表在软件工程领域的顶级会议或期刊上。我们对每年在以上会议或期刊上发表的论文数目进行统计，如表 1 所列。表 1 中的数据说明，近年来，并发软件测试逐渐成为了研究热点。

表 1 并发软件测试论文数目的统计

Table 1 Statistic of number of papers on concurrent software test

年份	期刊			会议			合计
	TSE	TOPLAS	TOSEM	ICSE	ESEC/FSE	ASE	
2017	0	1	2	4	5	7	19
2016	5	1	1	6	7	6	26
2015	3	1	1	7	10	5	27
2014	3	4	0	7	7	2	23
2013	0	1	3	4	6	3	17
2012	2	0	0	4	4	1	11
2011	0	1	0	7	1	3	12
2010	1	2	0	8	7	0	18
2009	1	0	0	6	5	2	14
2008	0	2	1	2	3	2	10
2007	0	0	0	4	3	4	11
2006	2	3	0	4	1	1	11

本文重点对已有的并发缺陷检测工作进行分析。具体来说，并发缺陷检测通过静态分析、动态分析或动静结合的方式完成。在缺陷检测的基础上，可开展并发缺陷的重现、诊断和修复工作。为了深入调研并发缺陷检测方法的研究进展，我们查阅表 1 中所有文献并筛选出 94 篇相关论文，然后按研究方法将其进行归类。我们发现，动态分析方法较多，常采用执行控制技术（18 篇）快速暴露并发缺陷；在静态方法中，数据流分析技术、符号执行和模型检测占主流；动静结合的研究相对较少。

本文第 2 节对并发缺陷进行分类，并给出相应定义；第 3 节对已有的并发缺陷检测技术进行分类、分析和比较；第 4 节对并发缺陷检测技术的通用性进行分析；第 5 节展望未来的研究重点；最后总结全文。

2 并发缺陷的分类

并发缺陷指两个或两个以上的线程因交互地作用于一个或多个共享资源而引起的程序崩溃或挂起，或产生与串行执行不同的结果，且这样的结果是不确定的。

我们发现，近年来也不乏对高级程序语义相关的并发缺陷的研究，如并发类型状态缺陷。因此，本文将并发缺陷分为并发类型状态缺陷、死锁、数据竞争、原子性违背和顺序违背。

定义 1（并发类型状态缺陷） 若线程的类型状态发生改变，且没有与另外线程的其他操作同步，则会造成对象执行其类型状态中不允许的操作，从而产生并发类型状态缺陷。

定义 2（死锁） 两个或两个以上的进程/线程在执行过程中，由于竞争资源或彼此通信而无限期地陷入僵持的局面，如果无外力作用，它们都将无法推进下去，此时称该系统产生了死锁。由于竞争资源而产生的死锁称为资源死锁；由于接收不到其他线程发来的消息而陷入无限等待产生的死锁称为通信死锁。

定义 3（数据竞争） 两个或两个以上的线程同时访问相同的内存单元，且至少有一个访问是写操作，同时线程间没有时序约束。

定义 4（原子性违背） 对共享内存的多个访问存在交错执行，打破了期望的顺序执行序列，即某个代码区域应该是原子的，但在执行过程中因交错执行破坏了其原子性，从而产生与该指令序列原子地执行时不同的执行效果。

定义 5（顺序违背） 两个内存访问没有按照期望的顺序执行，导致程序异常甚至崩溃。

3 并发缺陷检测技术

在对已有的研究工作进行系统总结的基础上，根据并发缺陷检测技术的思想手段的不同，将并发缺陷检测分为静态分析、动态分析、动静结合 3 类，如图 1 所示。早期研究主要集中于静态分析，后来随着对并发缺陷研究的不断深入，动态分析和动静结合的方法逐渐成为主导。

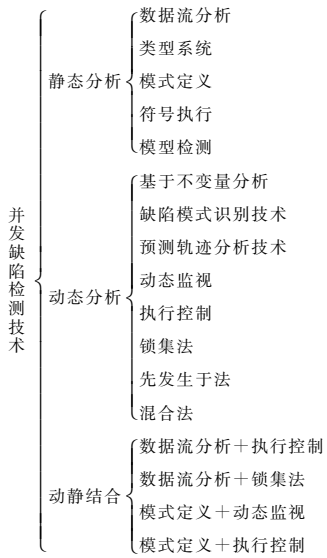

图 1 并发缺陷检测技术的分类

Fig. 1 Classification of concurrency bug detection techniques

3.1 静态分析

静态分析以人为定义的缺陷模式为指导，对分析文件进行模式提取和对比，以检测缺陷。静态分析具有操作简单、计算量相对较小等特点，但思想保守，误报率较高，需要较多的人工干预。静态分析方法可以进一步分为数据流分析、类型系统、模式定义、符号执行和模型检测 5 类。

1）数据流分析技术。该技术直接分析程序源代码，通过指向分析和逃逸分析计算共享变量、多线程访问共享变量的位置，并判断不同锁集中的元素是否指向同一个锁对象，从而检测数据竞争、死锁和原子性违背等并发缺陷；或计算程序的静态锁顺序图，将图中的环报告为可能的死锁。

Jlint[11]是最早使用锁顺序图检测死锁的工具，但只实现了对同步方法的分析。随后，Artho 和 Biere 对 Jlint 进行了扩展，提出了 Jlint2[12]，基于数据流分析检查不同类中的方法

或同步块是否占有相同的锁，以检测潜在的死锁；但未进行别名分析，存在较多的漏报。针对以上问题，Naik 等[13]通过对象敏感分析构造调用图和指向关系，通过别名分析、逃逸分析、并行分析排除不可能发生的死锁。RacerX[14]使用流敏感的过程间静态分析检测数据竞争和死锁，具有较好的可扩展性和准确性，但无法保证检测到所有的竞争。Chord[15]使用线程逃逸分析计算逃逸对，静态检测数据竞争。MUVI[16]通过流不敏感的和过程间静态数据流分析收集变量的访问信息，进而推断多变量原子性违背。随后，Huang[17]指出逃逸分析的一些局限性，并基于指向分析和调用图设计了可扩展的线程共享分析算法，该算法更加精确且快速地识别出了大型并发程序中线程访问共享数据的位置。

2)类型系统。类型系统将缺陷检测问题转化为类型检查问题，即检查所有对共享变量的访问是否都有一致性锁保护。若没有，则该访问类型为非法类型，可能会构成数据竞争等并发缺陷。早期的类型系统方法[18-19]需要开发人员添加大量注释，或标注原子集标签。Flanagan 等[20]使类型系统支持线程本地对象和参数化的类及方法，且扩展到大型系统，识别并检查并发程序中的原子属性。它可以为程序中未注释的方法自动且精确地推断原子性，减轻了编程人员的注释负担。

3)模式定义。模式定义是指人工指定或预先定义好模式，然后通过静态分析自动推断原子性区域、原子变量集或错误交互点。该方法主要针对非死锁缺陷检测，具有漏报率较低的优势，但是扩展性较差。FindBugs[21-22]是模式定义方法较早的且最为典型的应用工具，其收集了 18 种缺陷模式，如空指针异常、双重检查锁、同步不一致等，然后将 Java 字节码与预先定义的缺陷模式进行匹配来检测缺陷。随后，Vaziri 等[23]根据 11 个有问题的交互场景对数据竞争进行重新定义，构成一套完整的非序列化模式，并通过静态分析自动推断代码中需要同步的位置，以检测数据竞争。

4)符号执行。符号执行是一种使用符号值代替具体值来执行程序的技术，是一种基于路径的程序分析方法，最早提出于 20 世纪 70 年代[24]。它使用数学和逻辑符号对程序进行抽象，具有普遍性和严格性。在分析过程中，路径条件表示为关于符号的取值约束，路径条件的可满足问题则转化为约束可满足问题。现代符号执行技术结合具体值和符号值来有效探测程序的所有路径。Concolic Testing[25-26]最具代表性，却只适用于串行程序。Sen[27]最早设计并实现了并发测试工具 jCUTE，专门用于测试多线程 Java 程序。Con2colic[28]使 Concolic Testing 更加一般化，使用符号执行获得输入和调度序列，能更加系统地探索并发程序的执行空间。

符号执行是一种精确的程序分析方法，不存在误报[29]。然而，著名的路径爆炸问题使其扩展性较差。针对该问题，研究人员提出了各种缓解和解决办法，如使用摘要[30]、路径合并[31]、启发式搜索[32]等。

5)模型检测。模型检测是一种验证有限状态系统时序逻辑属性的形式化方法。首先将软件构造为状态机等抽象模型，然后使用模态/时序逻辑公式等形式化的表达式来描述安全属性，最终对模型进行遍历以验证软件是否满足安全属性，从而判断是否存在缺陷。Verisoft[33]是基于程序执行信息提取的模型检测工具，通过调度程序执行，利用程序执行信息(堆、栈和寄存器等)构造状态转换图，然后进行属性验证。此后，Verisoft，JPF[34]和 SPIN[35]模型检测工具得到了广泛应用。

尽管模型检测具有以上优势，但是与符号执行一样，状态空间爆炸问题是一大挑战。针对这一问题，学术界已经提出一些解决方法，通过减少和压缩状态空间来降低计算负载，如偏序规约技术[36]、最大因果规约技术[37-39]、有界模型检测[40-41]和抽象技术[42-43]。

3.2　动态分析

动态测试通过动态地执行被测程序，以期能够触发程序异常，并且触发的异常都是真实存在的问题，人工干预度较低，因此误报率较低；但并发交错执行的不确定性，导致其漏报率比较高。动态分析方法可以进一步分为基于不变量分析技术、缺陷模式识别技术、预测轨迹分析技术、动态监视技术、执行控制技术、锁集法、先发生于法和混合法 8 类。

1)基于不变量分析技术。不变量是程序正确执行期间维持的某些性质，反映了编程人员对程序行为的意图。不变量传统上被用于编译优化，近几年，研究人员通过训练执行程序推断可能的不变量，检测并发缺陷。早期，Liblit[44]通过统计部署软件中执行的多样性来探索基于不变量的缺陷检测。根据统计结果，将程序不变量分为 3 类：基于特征的不变量、基于关系的不变量和基于值的不变量。随后，Shi 等[45]提出一类新的程序不变量——定义使用不变量，捕获定义引用间的内在联系，并实现了工具 DefUse，其不仅可以检测多种常见的并发缺陷，还能够检测内存缺陷和语义缺陷。最近，Zhang 等[46]提出预期不变量，以及时检测多种类型的并发缺陷(包括原子性违背和顺序违背)。

2)缺陷模式识别技术。缺陷模式识别是根据预先定义的缺陷模式动态推断识别原子性区域或不可序列化的交错执行。该技术具有人工干预少、扩展性好等优点，但漏报率比较高。早期的缺陷模式识别技术大多只针对原子性违背模式。Velodrome[47]是第一个可靠且完备的原子性违背动态分析工具。为了有效应对并发程序测试中庞大的交互空间问题，Penelop[48]预先定义了 5 种典型的原子性缺陷模式，然后只选择会导致这些缺陷模式的调度进行重新执行，使得检测过程更加高效。随后，Park 等开发出的 Flacon[49]成为了第一个能同时检测原子性违背和顺序违背的动态缺陷定位工具。该工具结合缺陷定位技术，检测程序成功/失败执行的数据访问模式，报告具有不同可疑度的数据访问模式，并将其进行排序。实验验证了该工具的有效性和高效性。之后，他们对 Flacon 进行扩展，提出 Unicorn[50]，以识别并检测造成非死锁并发缺陷的数据访问模式。

3)预测轨迹分析技术。预测轨迹分析技术是为了更早地检测软件缺陷而产生的一种分析技术。首先记录一条程序的正常执行轨迹，然后在满足某些调度约束的前提下将路径中的事件重排列，以检测未暴露的并发缺陷。该技术可以预测并揭露未经历过的执行路径中的并发缺陷，因而漏报率较低；此外，该技术基于动态收集到的路径进行分析，因而误报率较低。然而，状态空间爆炸问题使得该技术难以扩展到较长轨迹。

预测轨迹分析技术的思想最初源于 Wang 等[51]提出的一种原子性缺陷检测方法。近年来,该技术受到了国内外研究学者的广泛关注,Huang 等[52-55]对此做出的贡献尤为突出。例如,他们开发了 PECAN[52]来预测多种并发访问异常,对于每个预测的异常访问,PECAN 可以生成具体的执行路径来确定性地暴露程度异常;设计了最大可靠的动态数据竞争检测工具 RVPredict[38],该工具能检测到许多之前从未发现的数据竞争;经过进一步推广,实现了 GPredict[54],其能检测更一般化的高级并发特性违背。

4)动态监视技术。动态监视技术在程序运行时监视多线程的交错情况,识别共享变量和交错访问等信息。该技术的误报率低,但程序运行的不确定性导致了漏报率非常高。Biswas 等[56]提出双重检查的方式来检测原子性违背,历经 2 次相互协作的动态分析。第一次运行完全跟踪线程间的依赖,虽然不精确,但提高了性能;第二次是精确分析,只处理第一次分析时识别为可能具有原子性违背的部分,保证了精度。

5)执行控制技术。执行控制技术在程序运行时控制线程调度,使得并发缺陷容易暴露出来。该技术增大了缺陷暴露的概率,因此漏报率较低,但执行开销较高。早期,Edelstein 等[57]提出在同步点注入人工延迟[58]以增大竞争同步资源的概率,并实现了工具 ConTest。ConTest 有助于暴露死锁,但是在暴露原子性违背方面的效率并不高。随后,RaceFuzzer[59]基于竞争条件,随机控制竞争指令的执行顺序,降低了误报率。但是,它暴露缺陷的能力依赖于竞争检测器,而大多数的数据竞争检测器很难达到较高的覆盖率。针对以上问题,Park 等[60]提出 CTrigger,利用最小执行扰动控制线程的调度,以增大原子性违背的暴露概率。

6)锁集法。锁集法检查多个线程对同一个共享变量的访问是否持有相同锁。如果没有共同锁保护共享资源,则可能因交错执行而产生并发缺陷。在给定精确信息的情况下,锁集法可以检测多种并发缺陷。但是,该方法忽略了除锁之外的其他同步原语(如 signal/wait/fork/join 等),因而会有较高的误报率。早期,Eraser[61]引入锁集算法推断同步,动态检测数据竞争,但不能用于更准确的同步模式。Atomizer[62]结合锁集算法和 Lipton 规约理论识别具备原子性的程序代码块,是第一个自动检测原子性违背的工具,但受锁集理论的限制,其识别精度并不高。Goldilocks[63]扩展了锁集的概念,构造同步设施集合和线程集合,是一个精确的竞争检测器,但执行效率并不高。

7)先发生于法。先发生于法基于 Lamport 的先发生于关系[64]。如果 2 个线程同时访问一个共享内存,且是因果无序的,则认为可能会发生竞争。先发生于法没有误报,然而,由于交互敏感性,其检测能力弱于锁集法,会出现漏报。此外,该方法实现比较困难,扩展性较差。传统的先发生于法基于时钟向量技术,如 DJIT 算法[65],只针对单个操作,且需要较高的时间和空间复杂度。随后,Flanagan 等提出 Velodrome[47],在程序执行中推断各种操作之间的先发生于边,然后构造事务的先发生于关系图,有环出现则表明观察的路径是不可串行化的。此外,他们还使用一种轻量级的自适应表示方法替代传统的时钟向量技术,实现了数据竞争检测器 FastTrack[66]。最近,Cai 等[67]提出时钟竞争的概念,并借助硬件支持进行竞争采样,降低了动态检测的运行负载。

8)混合法。混合法结合两种或两种以上方法的优点,以达到更好的检测效果,误报率和漏报率都相对较低。如 ASSETFUZZER[68]使用锁集法和先发生于法识别数据竞争和原子性违背。如 Velodrome[47]在动态监视程序执行时构造事务的先发生于关系图,描述事务间的依赖关系,保证了可靠性和较高的精度。

3.3 动静结合

动静结合的分析技术结合了动态分析的准确性和静态分析的完备性。静态分析为动态分析运行时验证减少了负载;动态分析为静态分析提供了可靠性保证。

数据流分析和执行控制技术结合,可以有效提高方法的精度,降低方法的误报率和漏报率。同时,数据流分析为后期执行控制技术指引了方向,可以大大缩短并发缺陷检测的时间,提高方法的性能。例如,ConSeq[69]静态识别字节码程序中可能的失败点,使用静态切片识别控制依赖和数据依赖影响失败点的关键读指令,然后在程序执行中控制线程调度以识别导致软件失败的可疑交互。最近,Cai 等[70]提出一种基于约束的方法来执行线程调度,以验证死锁。对于已确认的死锁,通过静态数据流分析识别导致该死锁发生的冲突内存访问,有利于开发人员理解并修复死锁。

数据流分析和锁集法结合,有利于降低漏报率,提高方法的性能,但仍有较高的误报率。例如,Choi 等[71]结合静态分析和动态锁集法改进了 Eraser,利用数据流分析识别可能参与数据竞争的内存访问语句,移除运行时不必要的检查,降低了运行时负载;然后通过插装和运行时优化进一步识别和丢弃冗余的访问事件;最后运行时检测器在程序运行时利用优化后的锁集法检查访问事件,进而检测数据竞争。Yoga 等[72]结合数据流分析和锁集法并行检测程序中的数据竞争,并通过实验验证了方法的有效性和高效性。

模式定义和动态监视技术相结合具有较高的人工干预度,可扩展性较差,且具有较高的运行负载;但是,该方法可以在一定程度上降低误报率。例如,Hammer 等[73]最先基于文献中定义的一系列有问题的访问模式提出一种运行时动态监视技术,自动检测原子集序列化违背,包括数据竞争和原子性违背。Lucia 等[74]从简化分析复杂度的角度,将共享变量形象化为颜色,同一类原子变量集中的变量具有相同的颜色,对变量的访问则转变为对某种颜色的访问。他们预先定义了 5 种类型的不可序列化交互模式,然后执行程序,线上检测单(多)变量原子性违背。

模式定义结合执行控制技术,首先预指定检测的模式类型,静态识别并推断可能的并发错误交互点,然后添加随机延时扰动控制线程执行,以检测并发缺陷。例如,AtomRace[75]利用模式定义识别原子的或交互访问的语句块,在其前后进行插装,为后期执行控制技术打下基础。同时,在动态分析过程中添加延时扰动,使得检测过程覆盖更多的访问交互。

4 并发缺陷检测技术的通用性分析

本节总结不同的并发缺陷检测技术与并发缺陷类型之间

的关系。表 2 列出了并发缺陷检测方法针对本文介绍的 5 类并发缺陷的适用性情况。

表 2　并发缺陷检测技术通用性统计

Table 2　Statistic of universality about concurrency bug detection techniques

思想	方法	并发缺陷				
		死锁	数据竞争	原子性违背	顺序违背	并发类型状态缺陷
静态分析	数据流分析	√	√	√		
	类型系统	√	√	√		
	模式定义		√	√		
	符号执行	√	√	√	√	√
	模型检测	√	√	√	√	
动态分析	基于不变量分析			√	√	
	缺陷模式识别技术			√	√	
	预测轨迹分析技术	√	√	√	√	√
	动态监视	√	√	√	√	
	执行控制	√	√	√		√
	锁集法		√	√		
	先发生于法		√	√		
	混合法		√	√		
动静结合	数据流分析＋执行控制	√	√	√	√	
	数据流分析＋锁集法		√	√		
	模式定义＋动态监视		√	√		
	模式定义＋执行控制		√	√	√	√

从表 2 中可以看出：

1）对高级并发缺陷中并发类型状态缺陷的研究方法较为单一。低级并发缺陷研究中，大多数方法集中在对数据竞争和原子性违背的检测，死锁和顺序违背次之。

2）符号执行和预测轨迹分析技术的通用性最好，可以检测 5 种并发缺陷，其次是模型检测和执行控制技术。

3）数据流分析技术主要用于检测死锁、数据竞争和原子性违背。模式定义方法主要用于检测底层内存访问中的非死锁并发缺陷。

4）并发缺陷检测研究中，动态分析被广泛应用。其中，针对数据竞争和原子性违背，常采用动态监视、执行控制技术、锁集法、先发生于法或混合法。

5）模型检测技术通过验证软件是否满足多种安全属性来判断是否存在缺陷，因此，也可以检测多种并发缺陷。但是，空间爆炸问题限制了该方法无法适用于大型并发应用程序中。

6）动静结合方法可以中和动态分析和静态分析的优缺点，多适用于数据竞争和原子性违背。

5　研究展望

目前，虽然并发缺陷检测方法已经取得众多实质性的研究成果，但该研究领域还存在大量值得国内外研究人员进一步关注的研究问题。

1）更加通用、准确的并发缺陷检测。并发软件可能存在不止一种并发缺陷，特别是对于大型并发程序，如开源数据库 MySQL。已有的并发缺陷检测工具的通用性较差，测试人员必须使用多种缺陷检测工具对同一个软件进行多次检测。开发更加通用、准确的并发缺陷检测工具，有利于提高测试效率。

2）软硬件相结合的并发缺陷检测。纯软件方法动态检测并发缺陷对硬件平台没有特殊要求，适用性较强，但会降低并发程序的运行速度；纯硬件方法可以降低执行开销，但需要修改当前的体系结构，可用性较差，而且不同种类的并发缺陷需要设计不同的硬件和缓存协议。将来的研究应设计软硬件结合的并发缺陷检测工具，以充分利用软件和硬件技术的优势来检测多种并发缺陷。

3）并发缺陷检测及修复[76]一体化。已有研究多集中于单独检测或单独修复并发缺陷。然而，有的修复工具依赖检测工具的报告结果，检测工具出现漏报时会导致错过修复，出现误报时则会继续错误地修复，从而对后期造成麻烦。将来的研究可以设计更多工具同时检测、诊断并修复所有并发缺陷。

4）适用于更加松散的内存模型的并发缺陷检测。顺序一致性模型过于理想化，抑制了在硬件和编译器上的一些性能优化功能。目前在松散内存模型下的并发缺陷检测技术较少[77-78]，导致已有的并发缺陷检测工作难以在实际中应用。设计并开发更多适用于弱内存模型的并发缺陷检测工具，将是学术界和工业界深入研究的一个重大问题。

5）推广在安卓等其他应用平台的并发缺陷研究[79-80]。宽带无线接入技术和移动终端技术的飞速发展，促使人类进入移动互联网时代。安卓平台是事件驱动并基于多线程架构的，且需要随时监听多种类型的事件[81]。在该平台下的并发缺陷检测会面临更大的挑战。

6）深入对分布式系统非确定性并发缺陷的研究。目前，云计算[82]发展如火如荼，分布式软件已超越了单机软件，成为主导领域。然而，以数据为中心的分布式系统的可靠性遭受着非确定性并发缺陷的严重威胁[83]。如何将研究成果合理地应用到对分布式并发缺陷的研究中，并探索新的研究方法进行分布式系统非确定性并发缺陷的检测、诊断和修复，将是未来需要关注的一个重要研究问题。

结束语　本文对已有的并发缺陷检测研究进行系统的分类和总结。首先，从研究背景展开介绍，将目前常见的并发缺陷分为并发类型状态缺陷、死锁、数据竞争、原子性违背和顺序违背五大类；然后，对现有的并发缺陷检测技术进行分类比较、分析和总结，根据方法手段的不同将并发缺陷检测技术分为静态分析、动态分析、动静结合，并从有效性、高效性、扩展性和人工干预度等方面做出评价，同时系统分析并总结了以上技术对 5 种并发缺陷的检测能力；最后，提出一些在并发缺陷检测研究中仍值得我们进一步探讨的研究问题。

参 考 文 献

[1] PINTO G, TORRES W, FERNANDES B, et al. A large-scale study on the usage of Java's concurrent programming constructs [J]. Journal of Systems & Software, 2015, 106(C): 59-81.

[2] LEVESON N G, TURNER C S. An investigation of the Therac-25 accidents [J]. Computer, 1993, 26(7): 18-41.

[3] POULSEN K. Software bug contributed to blackout [EB/OL]. http://www.securityfocus.com/news/8016.

[4] TAYLOR R N. A general-purpose algorithm for analyzing concurrent programs [J]. Communications of the ACM, 1983, 26(5): 361-376.

[5] TAYLOR R N, LEVINE D L, KELLY C D. Structural Testing

of Concurrent Programs [J]. IEEE Transactions on Software Engineering, 1992, 18(3): 206-215.

[6] SU X H, YU Z, WANG T T, et al. An Survey on Exposing, Detecting and Avoiding Concurrency Bug [J]. Chinese Journal of Computers, 2015, 38(11): 2215-2233. (in Chinese)
苏小红, 禹振, 王甜甜, 等. 并发缺陷暴露、检测与规避研究综述 [J]. 计算机学报, 2015, 38(11): 2215-2233.

[7] JIANG Y Y, XU C, MA X X, et al. Approaches to Obtaining Shared Memory Dependences for Dynamic Analysis of Concurrent Programs: A Survey [J]. Ruan Journal of Software, 2017, 28(4): 747-763. (in Chinese)
蒋炎岩, 许畅, 马晓星, 等. 获取访存依赖: 并发程序动态分析基础技术综述 [J]. 软件学报, 2017, 28(4): 747-763.

[8] LU S, PARK S, SEO E, et al. Learning from Mistakes: A Comprehensive Study on Real World concurrency bug characteristics [C]// Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2008: 329-339.

[9] WU Z D, LU K, WANG X P. Surveying concurrency bug detectors based on types of detected bugs [J]. Science China Information Sciences, 2017(3): 5-31.

[10] BIANCHI F A, MARGARA A, PEZZE M. A Survey of Recent Trends in Testing Concurrent Software Systems [J]. IEEE Transactions on Software Engineering, 2017, PP(99): 1-40.

[11] Jlint1[EB/OL]. http://www.ispras.ru/~knizhnik/jlint.

[12] ARTHO C, BIERE A. Applying Static Analysis to Large-Scale, Multi-Threaded Java Programs[C]// Proceedings of the 13th Australian Conference on Software Engineering. Washington: IEEE, 2001: 68.

[13] NAIK M, PARK C S, SEN K, et al. Effective Static Deadlock Detection[C]// Proceedings of the 31st International Conference on Software Engineering. Washington: IEEE, 2009: 386-396.

[14] 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.

[15] NAIK M, AIKEN A, WHALEY J. Effective static race detection for Java [J]. Acm Sigplan Notices, 2006, 41(6): 308-319.

[16] LU S, PARK S, HU C, et al. MUVI: Automatically Inferring Multi-variable Access Correlations and Detecting Related Semantic and Concurrency Bugs[C]// Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles. New York: ACM, 2007: 103-116.

[17] HUANG J. Scalable Thread Sharing Analysis[C]// Proceedings of the International Conference on Software Engineering. New York: ACM, 2016: 1097-1108.

[18] STERLING N. Warlock-A Static Data Race Analysis Tool[C]// Proceedings of the USENlx Winter. San Diego, 1993: 97-106.

[19] SASTURKAR A, AGARWAL R, WANG L, et al. Automated Type-Based Analysis of Data Races and Atomicity[C]// Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM, 2005: 83-94.

[20] FLANAGAN C, FREUND S N, LIFSHIN M, et al. Types for atomicity: Static checking and inference for Java [J]. ACM Trans on Programming Languages & Systems, 2008, 30(4): 1-53.

[21] HOVEMEYER D, PUGH W. Finding bugs is easy [J]. Acm Sigplan Notices, 2004, 39(12): 132-136.

[22] HOVEMEYER D, PUGH W. Finding Concurrency Bugs In Java [C]// Proceedings of the PODC Workshop on Concurrency & Synchronization in Java Programs. 2004: 1-10.

[23] VAZIRI M, TIP F, DOLBY J. Associating synchronization constraints with data in an object-oriented language[C]// Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York: ACM, 2006: 334-345.

[24] KING J C. Symbolic execution and program testing [J]. Communications of the ACM, 1976, 19(7): 385-394.

[25] SEN K. Concolic Testing[C]// Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering. New York: ACM, 2007: 571-572.

[26] WANG X Y, SUN J, CHEN Z B, et al. Towards optimal concolic testing[C]// Proceedings of the 40th International Conference on Software Engineering. New York: ACM, 2018: 291-302.

[27] SEN K. Scalable automated methods for dynamic program analysis[D]. Illinois: University of Illinois at Urbana-Champaign, 2006.

[28] FARZAN A, HOLZER A, RAZAVI N, et al. Con2colic Testing [C]// Proceedings of the 9th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2013: 37-47.

[29] YU T T, ZAMAN T S, WANG C. DESCRY: reproducing system-level concurrency failures[C]// Proceedings of the 11th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2017: 694-704.

[30] QADEER S, RAJAMANI S K, REHOF J. Summarizing Procedures in Concurrent Programs[C]// Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of programming languages. New York: ACM, 2004: 245-255.

[31] KUZNETSOV V, KINDER J, BUCUR S, et al. Efficient state merging in symbolic execution [J]. Acm Sigplan Notices, 2012, 47(6): 193-204.

[32] GUO S, KUSANO M, WANG C, et al. Assertion guided symbolic execution of multithreaded programs[C]// Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2015: 854-865.

[33] GODEFROID P. Model Checking for Programming Languages using VeriSoft[C]// Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of programming languages. New York: ACM, 1997: 174-186.

[34] BRAT G, HAVELUND K, VISSER W. Java PathFinder-Second Generation of a Java Model Checker[C]// Proceedings of the Workshop on Advances in Verification. 2000: 1-6.

[35] HOLZMANN G J. The Model Checker SPIN [J]. IEEE Transactions on Software Engineering, 2002, 23(5): 279-295.

[36] 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.

[37] SERBANUTA T F，CHEN F，ROSU G. Maximal Causal Models for Sequentially Consistent Systems[C]// Proceedings of the International Conference on Runtime Verification. Berlin：Springer，2012：136-150.

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

[39] HUANG J. Stateless model checking concurrent programs with maximal causality reduction [J]. Acm Sigplan Notices，2015，50(6)：165-174.

[40] WU X，WEN Y，CHEN L，et al. Data Race Detection for Interrupt-Driven Programs via Bounded Model Checking[C]// Proceedings of the IEEE 7th International Conference on Software Security and Reliability-Companion. Washington：IEEE，2013：204-210.

[41] KHOSHNOOD S，KUSANO M，WANG C. ConcBugAssist：Constraint Solving for Diagnosis and Repair of Concurrency Bugs[C]// Proceedings of the International Symposium on Software Testing and Analysis. New York：ACM，2015：165-176.

[42] PATRICK C，RADHIA C. Abstract Interpretation Frameworks [J]. Journal of Logic & Computation，1992，2(4)：511-547.

[43] MCMILLAN K L. Lazy Abstraction With Interpolants[C]// Proceedings of the International Conference on Computer Aided Verification. Berlin：Springer，2006：123-136.

[44] LIBLIT B. Cooperative Bug Isolation [M]. Berlin：Springer，2004：255-264.

[45] SHI Y，PARK S，YIN Z，et al. Do I Use the Wrong Definition? DeFuse：Definition-Use Invariants for Detecting Concurrency and Sequential Bugs[C]// Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications. New York：ACM，2010：160-174.

[46] ZHANG M，WU Y，LU S，et al. A Lightweight System for Detecting and Tolerating Concurrency Bugs [J]. IEEE Trans on Software Engineering，2016，42(10)：899-917.

[47] FLANAGAN C，FREUND S N，YI J. Velodrome：a sound and complete dynamic atomicity checker for multithreaded programs [C]// Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. New York：ACM，2008：293-303.

[48] SORRENTINO F，FARZAN A，MADHUSUDAN P. PENELOPE：Weaving Threads to Expose Atomicity Violations[C]// Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York：ACM，2010：37-46.

[49] PARK S，VUDUC R W，HARROLD M J. Falcon：Fault Localization in Concurrent Programs[C]// Proceedings of the 32nd International Conference of Software Engineering. New York：ACM，2010：245-254.

[50] PARK S，VUDUC R，HARROLD M J. UNICORN：a unified approach for localizing non-deadlock concurrency bugs [J]. Software Testing，Verification & Reliability，2015，25(3)：167-190.

[51] WANG L，STOLLER S D. Accurate and Efficient Runtime Detection of Atomicity Errors in Concurrent Programs[C]// Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York：ACM，2006：137-146.

[52] HUANG J，ZHANG C. Persuasive Prediction of Concurrency Access Anomalies[C]// Proceedings of the 2011 International Symposium on Software Testing and Analysis. New York：ACM，2011：144-154.

[53] HUANG J，RAJAGOPALAN A K. Precise and maximal race detection from incomplete traces [J]. Acm Sigplan Notices，2016，51(10)：462-476.

[54] HUANG J，LUO Q Z，ROSU G. GPredict：Generic Predictive Concurrency Analysis[C]// Proceedings of the 37th International Conference on Software Engineering. NJ：IEEE Press，2015：847-857.

[55] HUANG J. UFO：predictive concurrency use-after-free detection [C]// Proceedings of the 40th International Conference on Software Engineering. New York：ACM，2018：609-619.

[56] BISWAS S，HUANG J，SENGUPTA A，et al. DoubleChecker：Efficient Sound and Precise Atomicity Checking[C]// Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York：ACM，2014：28-39.

[57] EDELSTEIN O，FARCHI E，NIR Y，et al. Multithreaded Java program test generation [J]. Ibm Systems Journal，2002，41(1)：111-125.

[58] LEE S，HWANG S，RYU S. All about activity injection：Threats，semantics，and detection [C] // Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. New York：ACM，2017：252-262.

[59] SEN K. Race directed random testing of concurrent programs [J]. Acm Sigplan Notices，2008，43(6)：11-21.

[60] PARK S，LU S，ZHOU Y Y. CTrigger：exposing atomicity violation bugs from their hiding places [J]. ACM SIGARCH Computer Architecture News，2009，37(1)：25-36.

[61] SAVAGE S，BURROWS M，NELSON G，et al. Eraser：a dynamic data race detector for multithreaded programs [J]. ACM Trans on Computer Systems，1997，15(4)：391-411.

[62] FLANAGAN C，FREUND S N. Atomizer：a dynamic atomicity checker for multithreaded programs [J]. Science of Computer Programming，2008，71(2)：89-109.

[63] ELMAS T，QADEER S，TASIRAN S. Goldilocks：a race and transaction-aware java runtime[C]// Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York：ACM，2007：245-255.

[64] LAMPORT L. Time，clocks，and the ordering of events in a distributed system [J]. Communications of the Acm，1978，21(7)：558-565.

[65] POZNIANSKY E，SCHUSTER A. MultiRace：efficient on-the-fly data race detection in multithreaded C++ programs [J].

Concurrency & Computation Practice & Experience, 2007, 19(3):327-340.

[66] FLANAGAN C, FREUND S N. FastTrack: Efficient and Precise Dynamic Race Detection[C]// Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design & Implementation. New York: ACM, 2009: 121-133.

[67] CAI Y, ZHANG J, CAO L W, et al. A deployable sampling strategy for data race detection[C]// Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM, 2016: 810-821.

[68] LAI Z, CHEUNG S C, CHAN W K. Detecting Atomic-Set Serializability Violations in Multithreaded Programs through Active Randomized Testing[C]// Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering. New York: ACM, 2010: 235-244.

[69] ZHANG W, LIM J, OLICHANDRAN R, et al. ConSeq: detecting concurrency bugs through sequential errors[C]// Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2011: 251-264.

[70] CAI Y, LU Q. Dynamic Testing for Deadlocks via Constraints [J]. IEEE Transactions on Software Engineering, 2016, 42(9): 825-842.

[71] CHOI J D, LEE K, LOGINOV A, et al. Efficient and Precise Datarace Detection for Multithreaded Object-Oriented Programs [C]// Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation. New York: ACM, 2002: 258-269.

[72] YOGA A, NAGARAKATTE S, GUPTA A. Parallel Data Race Detection for Task Parallel Programs with Locks[C]// Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM, 2016: 833-845.

[73] HAMMER C, DOLBY J, VAZIRI M, et al. Dynamic Detection of Atomic-Set-Serializability Violations[C]// Proceedings of the 30th International Conference on Software Engineering. New York: ACM, 2008: 231-240.

[74] LUCIA B, CEZE L, STRAUSS K. ColorSafe: Architectural Support for Debugging and Dynamically Avoiding Multi-Variable Atomicity Violations[C]// Proceedings of the 37th Annual International Symposium on Computer architecture. Saint-Malo, New York: ACM, 2010: 222-233.

[75] LETKO Z, VOJNAR T, KRENA B. AtomRace: Data Race and Atomicity Violation Detector and Healer[C]// Proceedings of the 6th workshop on Parallel and distributed systems: testing, analysis, and debugging. New York: ACM, 2008: 1-10.

[76] CAI Y, CAO L W, ZHAO J. Adaptively Generating High Quality Fixes for Atomicity Violations[C]// Proceedings of the 11th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2017: 303-314.

[77] HUANG A. Maximally Stateless Model Checking for Concurrent Bugs under Relaxed Memory Models[C]// Proceedings of the IEEE/ACM 38th International Conference on Software Engineering Companion. New York: ACM, 2016: 686-688.

[78] KARTIK N, SURESH J. Automated Detection of Serializability Violations Under Weak Consistency[C]// Proceedings of the 29th International Conference on Concurrency Theory. Schloss Dagstuhl, 2018: 1-18.

[79] WANG J, JIANG Y Y, XU C, et al. AATT+: Effectively manifesting concurrency bugs in Android apps [J]. Science of Computer Programming, 2018, 163: 1-18.

[80] ZHOU J, SILVESTRO S, LIU H, et al. UNDEAD: Detecting and preventing deadlocks in production software[C]// Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. New York: ACM, 2017: 729-740.

[81] SUN Q. Detecting and Replaying Data Races in Android Applications [D]. Nanjing: Nanjing University, 2017. (in Chinese)
孙全. 安卓应用数据竞争的检测与再现 [D]. 南京: 南京大学, 2017.

[82] WIN T Y, TIANFIELD H, MAIR Q. Big Data Based Security Analytics for Protecting Virtualized Infrastructures in Cloud Computing [J]. IEEE Transactions on Big Data, 2018, 4(1): 11-25.

[83] LIU Y, SUN X H. CaL: Extending Data Locality to Consider Concurrency for Performance Optimization [J]. IEEE Transactions on Big Data, 2018, 5(2): 273-288.