# 并发程序数据竞争缺陷高效检测技术

## 1.研究目的及意义

随着多核处理器的普及和多核处理器的发展，越来越多的人开始使用并发编程来提高程序的性能。并发编程具有很多优势，它不仅可以减少程序的运行时间，而且可以提高程序的吞吐量和多核处理器的利用率。

虽然并发编程带了很多好处，但并发程序内部的并发性和不确定性仍然会导致一些难以避免的问题，包括死锁、数据竞争、原子性违背和顺序违背等， 这些并发问题都有着难以检测、调试和修复的特点。在这些并发问题中，数据竞争是指在多线程程序中，两个或多个线程在无时序限制情况下访问同一内存位置并且至少有一个线程执行写操作。数据竞争常常是引起其他非死锁并发缺陷的根本原因，并且在所有并发缺陷中占有较大比例[1]。

数据竞争[2]是多线程程序中常见的并发错误，它们中的许多可能会导致程序结果发生错误，更为严重地并发错误会直接导致程序和服务器崩溃，数据竞争是指对同一个共享内存空间，存在若干并发访问，并且至少有一个是写访问[2]，数据竞争不一定导致程序错误，因为有些程序员故意让程序有数据竞争以提高运行的效率，但是有调查表明 5%-24%的数据竞争会对程序产生坏影响[3]。在并发程序的安全性缺陷中，主要包括数据竞争、原子性违背[4]、顺序违背[5]以及死锁[6]。原子性违背是指原来必须原子性执行的指令序列，在并发交错的干扰下，其执行的效果不与任何原子性指令序列的执行效果相同，各个线程需保持的一致性遭到其他并发线程写操作破坏。顺序违背指的是一指令(组)没有按照预期执

行，总是在另一(组)指令之前或是之后执行。死锁指的是某个线程集合中每一个线程都在等待该集合中的另一个线程释放占有的互斥性资源，从而导致整体陷入循环等待状态。研究分析可知，数据竞争在上述常见的 4 种并发缺陷中占的比例较大[44]，并且大部分是导致原子性违背和顺序违背的根源。

在并发程序数据竞争的领域中，现有的检测方法很难同时做到高精度和高效率，大多数数据竞争发生在两条线程之间，也就是发生在两条来自不同线程的指令间，针对这一特性，对程序中所有来自不同线程指令两两之间形成的指令对进行分析可降低大量误报，同时引入 lockset 算法[8]以及 happens-before 关系[9]来降低误报，因此，本文将提出并实现了一种基于分类模型的并发程序数据竞争智能化检测技术，该技术利用对程序动态插桩得到的指令信息建立分类模型，并针对性的剔除隐型同步对，有效的解决上述问题。

## 2.国内外研究现状

国内外许多研究已经提出了很多数据竞争检测和验证的方法，主要分为三类: (1)静态数据竞争检测方法、(2)动态数据竞争检测方法、(3)动静结合的数据竞争验证方法，下面将对这三种验证方法的研究现状进行详细论述。

静态检测方法只需要分析程序源码，但是缺少程序运行时的信息，导致报告的数据竞争大部分都是误检。动态检测方法监视程序在执行过程中的行为，收集必要的信息来判断哪些访问操作构成数据竞争，但是受限于线程执行交错的不确定性以及收集到信息的不完整性，该方法依然会生成很多误检和漏检。动静结合的方法虽然能够弥补各自方法的一些缺陷，但是在源程序运行过程中引入了大量的性能开销，同时并没有真正显著提升数据竞争检测的精度。这些

验证方法虽然能够精准地找到数据竞争，但同时会造成大量的漏检并且验证效率也比较低。

静态数据竞争检测主要是进行锁集分析，从而检测数据竞争。常用的静态数据竞争检测工具包括 Warlock[10]、RacerX[11]、RELAY[12]和 Locksmith[13]。其中 RacerX 利用流敏感和过程间分析检测数据竞争和死锁；Locksmith 首先使用标签流约束和抽象控制流图约束信息来进行锁集分析，然后使用标签流约束、抽象控制流图约束和上下文敏感约束展开共享变量的分析，最后结合线性分析，检测出数据竞争；RELAY 通过控制流图和程序调用图，采用自底向上的分析方式，首先进行过程内锁集分析并缓存在函数标签中，然后开启过程间锁集分析，等到所有的线程相关的函数全部分析完毕，再判断相关的共享变量是否产生数据竞争。RELAY 由于其扩展性堪称优良，能够应用在百万级别代码量的程序上，因此，在实际使用过程中获得了高度认可与广泛接受。尽管静态数据竞争检测只需要分析程序源码、监测效率高并且基本不会有漏报，但由于其忽略了程序执行过程中的一些 happens-before 关系，因此静态数据竞争检测方法得到的绝大部分都不是真正的数据竞争。

现有的动态检测技术主要分为三种：基于 locket、基于 happens-before 与二者结合的方法。(1)基于 locket 的方法对线程交织不敏感，但是存在误报情况，即无效竞争。(2)基于 happens-before 的方法只检测某特定交织序列上的数据竞争，检测结果虽可靠，但敏感于线程交织。(3)混合方法结合了两者的优点，并且试图减少各自的缺点，但也面临如不能够搜索出隐藏的错误、locket 高误报引起的无效报警等问题。

## 2.1 国外研究现状

Savage 等[8]提出基于 lockset 的动态数据竞争检测方法 Eraser。该方法在程序执行过程中维护每个线程当前的锁集信息，同时更新共享变量持有的锁集信息，当共享变量不再受到锁保护的时候，报告出数据竞争。

Praun 和 Tayfun 等[14,15]在 Eraser 的基础上对基于 lockset 算法的动态数据竞争检测方法进行了精细和扩展，使得能够检测对象级别的数据竞争并且更加准确和有效。

Dinning、Mellor-Crummey 和 Perkovic 等[16-18]基于 Lamport 的 happens-before 关系提出了使用逻辑时钟来动态地检测数据竞争。Happens-before 关系是在多线程并发程序执行的所有事件上的一种偏序关系。该关系要求同一个线程内部按照时序逻辑顺序执行，而在线程间程序的执行依赖于同步机制。一旦对一个共享内存空间的访问违背了 happens-before 关系，那么就会产生数据竞争。

Pozniansky、Flanagan、Cai 以及 Ok-Kyoon 等[19-22]分别提出了改进后的基于 happens-before 关系的动态数据竞争检测方法。其中 Djit+算法使用 vector-clock 记录线程和共享内存空间访问的逻辑时钟，同时每一个时间帧中只记录第一次对共享内存空间的读/写访问。Fast Track 认为在一个没有数据竞争故障的程序执行过程中，对共享内存空间的写访问是有序的，而只有多个线程对同一共享内存空间进行读访问才可能是并发进行的。因此对于写访问可以采用轻量级的 epoch 形式的逻辑时钟，而只有并发的读访问才会依然采用 vector-clock 形式的逻辑时钟。Loft 在 Fast Track 的基础上提出一些场景来进一步减少基于 vector-clock 的拷贝和比较操作。iFT 对 Fast Track 进一步的简化，不需要遍历 vector-clock 的每一项进行分析，而只是关注 left-most 和 right-most 两项，将读

写访问操作的数据竞争检测复杂度降低到了 O(1)。

Pozniansky、Jannesari、Serebryany、Xinwei 和 Misun 等[19,23-31]分别提出了基于 hybrid 的动态数据竞争检测方法。Multi Race 首先利用改进的 lockset 算法找到可疑的数据竞争，然后利用 Djit+算法进一步检测可疑的语句对是否真正是并发执行的。Helgrind+将线程顺序执行的操作序列约束在一个 segment 中，segment 能够反映线程和线程的逻辑时钟。Helgrind+首先进行 happens-before 关系的分析找到所有潜在并发的 segment，然后利用 lockset 算法验证共享内存空间的访问是否被公共的锁保护。Thread Sanitizer 同样使用 segment 来表示一个线程中连续执行的内存访问事件。Segment 中第一个事件包含当前 segment 中所有事件的上下文信息，同时维护两个锁集合分别表示保护读/写持有的锁集合 LSrd 和 LSwr。Thread Sanitizer 维护两个 segment 集合，SSrd 和 SSwr 分别表示并发的读/写 segment 集合。任何对共享内存空间的访问都会更新并迭代遍历这两个 segment 集合，利用 lockset 算法检测数据竞争。Acculock 首先使用 epoch 形式的逻辑时钟进行 happens-before 关系的分析，然后再利用 lockset 算法验证。同时 Acculock 根据 lockset 的包含关系进一步的减少了冗余操作的分析。Multi Lock-HB 对 Acculock 中算法的固有缺陷进行了改进，提升了检测数据竞争的能力。

Marino 和 Bond 等[32,33]提出了基于抽样的动态数据竞争检测方法。Lite Race 是一种轻量级数据竞争检测算法，它对待检测的可疑的读写访问进行抽样分析，而不是针对所有可疑的读写访问。Pacer 在 Lite Race 的基础上改进了抽样部分的算法，提高了数据竞争产生的概率。

Xiong 和 Jannesari 等[25,34-36]提出了如何有效识别 ad-hoc 类型同步从而提升

数据竞争检测精度的方法。Ad-hoc 类型的同步样式很多，其中最常用的是自旋读（spinning read）和配对写（counterpart write）组合的形式。Helgrind+动态地识别可疑的自旋读，然后通过每个共享内存单元最后的写操作来确认这种隐式的同步关系成立。Sync Finder 则是通过静态分析来识别自旋读和远程写操作构成的 ad-hoc 类型同步关系。

Sen、Zhang、Kai 和 Kasikci 等[37-42]分别提出了动静结合的数据竞争检测方法。Race Fuzzer 根据静态检测出的可疑的数据竞争对，在动态执行的时候加入人工的干扰确保只有一个线程执行，从而验证哪些可疑的数据竞争对是有害的数据竞争。Con Mem，PRFinder，Col Finder 和 Race Checker 同样也是在 Race Fuzzer 的基础上进行了改进，使用延时阻塞代替确定性的阻塞，同时在验证数据竞争的过程中保证不相关的线程可以并发执行。Race Mob 核心的数据竞争检测和验证方法类似于前面 Race Fuzzer，但是它通过众包的方式按需在客户机上创建不同的程序环境，从而增加数据竞争产生条件的几率，验证数据竞争以及数据竞争的有害性。

## 2.2 国内研究现状

吴萍[46]等提出了一种准确有效的静态数据竞争检测方法。该方法使用比较准确的别名分析来静态地模拟事件序列。同时为了提高分析效率，该检测方法以对象为基准，结合逃逸分析来降低分析的复杂度。该方法在普通规模的程序上能够取得比较准确的分析结果，但是不能够区分错误和良性的数据竞争。

魏鹏[47]等提出了基于 lockset 算法的改进方法。该方法利用面向切面编程技术，在切入点检测待测程序的运行状态，利用线程间程序变量状态图和变量

集合来丰富 lockset 算法，从而达到更好的检测效果。

姚欣洪[48]等提出了一种静态数据竞争的方法。该方法基于抽象解释，采用线程摘要的方式描述每一个线程的方位行为，通过线程摘要的计算把并发程序转化为多个线程执行序列组成的模型，采用笛卡尔积的形式生成所有可疑的数据竞争语句对，最后根据四个不可能发生数据竞争的条件进行筛选，最终得到的集合就是可能发生数据竞争的语句对集合。

徐超[49]等提出了一种基于 hybrid 的动态数据竞争检测方法。该算法和之前提到的 Helgrind+核心思想基本一致，只是改进了共享变量的状态机模型。

# 3.主要研究内容及可行性分析

## 3.1 主要研究内容

(1) 利用 happens-before 关系与 lockset 算法动态指令级检测数据竞争

本课题首先利用动态二进制插桩工具 Pin 对被测程序进行动态二进制插桩，同时为每条线程创建一条向量时钟并对其进行跟踪，然后分析所有来自不同线程的指令两两之间形成的指令对，利用 happens-before 关系判断指令对间的向量时钟是否满足偏序关系，并用 lockset 算法判断指令对的锁集合是否为空，若不满足偏序关系且锁集合为空则报告数据竞争。lockset 算法对线程交错不太敏感，但不会考虑其他的一些同步原语，也就是无法剔除良性的数据竞争，所以导致很多的误报。若单纯的使用 happens-before 关系来检测数据竞争，则会对线程交错比较敏感，由于多线程程序各个线程调度的复杂性及随机性，会出现很多漏报。因此结合 lockset 算法和 happens-before 关系的混合算法由此而生，本课题也将基于该算法对多线程程序中可能产生的数据竞争进行动态检测。

(2) 剔除隐型同步对

lockset 算法和 happens-before 关系的混合算法仍然存在着一些缺陷，它无法识别一些常见的隐型同步对，导致产生一系列的误报，隐型同步对的种类繁多，常见的有配对写、自旋读以及隐形同步相关信息等，所以本课题将针对性的剔除一些常见的隐型同步对。

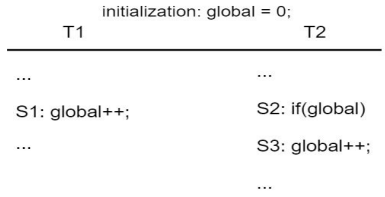


图 1 隐型同步对

(3) 利用分类模型分类预测数据竞争

本课题利用研究内容(1)中的动态二进制检测结果，进行特征提取，并建立分类模型进行数据分析，采用特定的分类算法进行，最后判断得到当前检测被检程序是否存在数据竞争，降低数据竞争检测漏报率和误报率，实现高效准确检测。

## 3.2 可行性分析

在编程语言方面，本人熟悉 C/C++语言，并在 C/C++方向做过一些项目实战,；在实验工具及平台方面，本人已经安装好实验环境以及熟悉掌握了工具该如何使用；在知识理论方面，本人已经读阅过比较多的与软件测试和人工智能领域相关的国内外参考文献，对软件测试以及人工智能领域涉及到的相关方法和技术背景有一定的理解和掌握，并熟悉现有的各种对多线程并发程序数据竞争检测的方法，有数据结构、编译原理、操作系统等学科的基础，掌握并能使用一些常用的算法，有对算法性能进行一定评价的能力；在实验能力方面，本

人具有通过网络、文献及自身知识等解决实际问题的能力，因此有充分的把握去解决攻克一些新的问题；本人会定期对近期研究工作进行总结并与指导老师进行研究与讨论，并且认真听取指导老师的意见，确保研究方向和过程的正确性和合理性。因此，本课题具备充分的可行性。

## 4.实验条件

本课题所涉及的实验是在 Pin 工具下进行的，Pin 是 Intel 发布的一款动态二进制插桩平台，它同时支持 IA-32、Intel64、IA-64CPU 架构以及 Windows、Unix/Linux 和 Mac OS 等操作系统，该平台具有功能强大、易用、可移植性好等特点。Pin 提供了简单易用且符合 C/C++编程语言规范的接口函数(API)，并且提供很多简单实用的示例供开发者参考。本课题采用的测试 BenchMark 是来自 google data-race-test 的一组公开多线程程序。本课题所有的实验都是在 Ubuntu18.04 上用 4 核处理器和 4GB 内存进行的。

## 5.预期达到的目标

(1) 成功开发多线程并发程序数据竞争检测框架，实现较低的误报率和漏报率；

(2) 申请国家发明专利一项；

(3) 发表一篇相关领域高水平学术论文；

# 6.难点及创新之处

## 6.1 难点

(1) 本课题采用分析所有来自不同线程的指令两两之间形成的指令对来检测数据竞争，虽然降低了大量的漏报，但这无疑会造成较大的开销，假若并发程序中有两条线程，每条线程有 $n$ 条指令，则该并发程序的指令对有 $n \times n$ 对，数据量过大。所以怎样降低开销成为本课题的难点之一。

(2) 隐型同步对是造成误报的一项重要因素，并且隐型同步对的种类繁多，而目前也没有较好的剔除方式，本课题中只是剔除了几种常见的隐型同步对，在实际情况中可能会出现更为复杂的隐型同步对，所以在今后的研究中，本课题将发掘并剔除更多的隐型同步对来提高检测精度。

(3) 在建立分类模型的过程中，数据预处理是一项关键环节，数据的形式必须根据各个分类器的需求而变化，并且在建立训练集的过程中，动态二进制插桩得到的数据也会存在一定的误差，在必要的情况下必须手动消除这些误差。

## 6.2 创新之处

(1) 目前还没有出现基于指令级的并发程序数据竞争检测技术，本课题利用 Pin 工具对程序动态二进制插桩可以得到程序中每条指令的信息，指令级的分析技术能消除大量的漏报，并利用 happens-before 关系以及 lockset 算法结合的方式进一步提高了检测精度。

(2) 现有的大多数并发程序数据竞争检测技术中都忽视了隐型同步对的影响，本课题将针对性的剔除隐型同步对来降低检测误报。

(3) 实现对应用程序的 Log 进行数据分析，通过提取特征变量，检测程序是否存在数据竞争缺陷，并提供相应的解决方案。

(4) 由于动态检测存在很大的开销，其中包括时间和内存上的消耗，故通过对 Pin 采集到的信息进行抽样，按照相关概率提取关键信息，然后在此基础上进行建模分析，可以大幅度降低开销。

# 7.主要问题及关键技术

## 7.1 主要问题

本课题所提出的并发程序数据竞争检测检测方法分为两个模块，分别为动态二进制插桩模块以及建立分类模型模块，而动态二进制插桩模块需要用到 Pin 工具，Pin 工具必须在 ubuntu 的环境下运行，无法实现分类模型与 Pin 框架的一体化，造成检测过程相对较为繁琐，在今后的工作中本课题将针对目前的问题继续完善并发程序数据竞争智能化检测技术，使其更加高效。

## 7.2 关键技术

本课题所涉及的实验首先将在 Pin 工具上开发动态数据竞争检测框架，利用 Pin 工具的指令级插桩效果，能够精准报告数据竞争及其位置。通过在采集到的信息上建立分类模型，准确判断是否存在数据竞争并及时反馈出结果。

图 2 描述了并发程序数据竞争智能化检测系统结构及数据竞争检测流程。该系统是由一个 Pin 二进制插桩框架和一个分类模型组成，Pin 框架对多线程程序进行动态插桩得到训练集和测试集，训练集需要检测数据竞争，而测试集只需打印指令信息即可，再将得到的训练集来训练分类模型，最后用训练好的模

型对测试集进行分类预测，报告出测试集中的数据竞争。

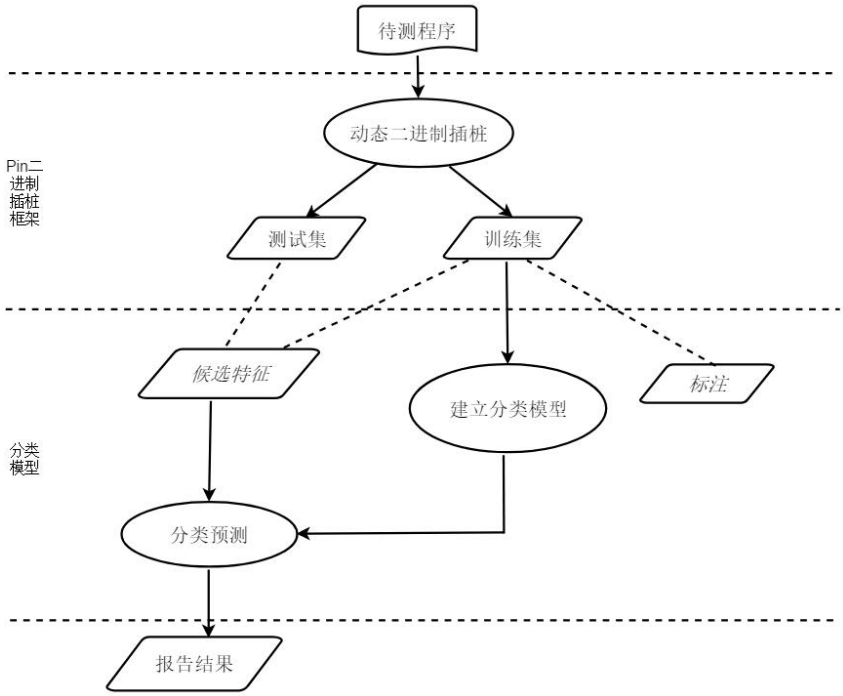


图 2 并发程序数据竞争智能化检测系统结构及流程图

# 8.研究计划进度表

| 起始日期 | 终止日期 | 研究内容 |
|---|---|---|
| 2019 年 9 月 1 日 | 2019 年 12 月 15 日 | 选题前期研究工作 |
| 2019 年 12 月 16 日 | 2019 年 12 月 23 日 | 开题报告及答辩 |
| 2019 年 12 月 24 日 | 2020 年 5 月 1 日 | 课题研究 |
| 2020 年 5 月 2 日 | 2020 年 9 月 1 日 | 论文写作 |
| 2020 年 9 月 2 日 | 2021 年 3 月 1 日 | 论文审核并准备答辩 |

# 9.参考文献


[1] Gochman S, Mendelson A, Naveh A, et al. Introduction to Intel core duo processor architecture [M]. Intel Technology Journal, 2006, 10(2): 89-97

[2] Netzer R H B, Miller B P. What are race conditions? Some issues and formalizations [J]. Acm Letters on Programming Languages & Systems, 1992, 1(1):74-88.

[3] Kasikci B, Zamfir C, Candea G. RaceMob: Crowd sourced data race detection [C]// Proc of SOSP'13, 2013:406-422.

[4] Praun C V, Gross T R. Static Detection of Atomicity Violations in Object-Oriented Programs [J]. Journal of Object Technology, 2003, 3(6):44-53.

[5] Engler D, Ashcraft K. Racer X: effective, static detection of race conditions and deadlocks [J]. ACM SIGOPS Operating Systems Review, 2003, 37(5):237-252.

[6] Lu S, Park S, Seo E, et al. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics [J]. Acm Sigarch Computer Architecture News, 2008, 36(3):329-339.

[7] Zhenlong Yuan, Yongqiang Lu, Yibo Xue. Droid Detector: Android Malware Characterization and Detection Using Deep Learning [J]. Tsinghua Science and Technology, 2016, 21(01): 114-123.

[8] Savage S, Burrows M, Nelson G, et al. Eraser: A dynamic data race detector for multithreaded programs [J]. ACM Transactions on Computer Systems (TOCS), 1997, 15(4): 391-411.

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

[10] Sterling N. Warlock - A static data race analysis tool[C]// 2010:97-106.

[11] Engler D, Ashcraft K. Racer X: effective, static detection of race conditions and deadlocks[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5):237-252.

[12] Voung J W, Jhala R, Lerner S. RELAY: static race detection on millions of lines of code[C]// Joint Meeting of the European Software Engineering Conference and the ACM Sigsoft International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September. 2007:205-214.

[13] Pratikakis P, Foster J S, Hicks M. LOCKSMITH: context-sensitive correlation analysis for race detection[C]// ACM Sigplan 2006 Conference on Programming Language Design and Implementation, Ottawa, Ontario, Canada, June. 2006:320-331.

[14] Von Praun C, Gross T R. Object race detection[C]//ACM SIGPLAN Notices. ACM, 2001, 36(11): 70-82.

[15] Elmas T, Qadeer S, Tasiran S. Goldilocks: Efficiently computing the happens-before relation using locksets[M]//Formal Approaches to Software Testing and Runtime Verification. Springer Berlin Heidelberg, 2006: 193-208.

[16] Dinning A, Schonberg E. An empirical comparison of monitoring algorithms for access anomaly detection[M]. ACM, 1990.

[17] Netzer R H B. Race condition detection for debugging shared-memory parallel programs[D]. UNIVERSITY OF WISCONSIN–MADISON, 1991.

[18] Perkovic D, Keleher P J. Online data-race detection via coherency guarantees[C]//OSDI.

1996, 96: 47-57.
[19] Pozniansky E, Schuster A. Efficient on-the-fly data race detection in multithreaded C++ programs[M]. ACM, 2003.
[20] Flanagan C, Freund S N. Fast Track: efficient and precise dynamic race detection[C]//ACM Sigplan Notices. ACM, 2009, 44(6): 121-133.
[21] Cai Y, Chan W K. LOFT: redundant synchronization event removal for data race detection[C]//Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on. IEEE, 2011: 160-169.
[22] Ha O K, Jun Y K. An Efficient Algorithm for On-the-Fly Data Race Detection Using an Epoch-Based Technique[J]. Scientific Programming, 2015, 2015(3):660-674.
[23] Valgrind-project., "Helgrind: a data-race detector," 2007. [Online]. Available: http://valgrind.org/docs/manual/hgmanual. Html
[24] Jannesari A, Tichy W F. On-the-fly race detection in multi-threaded programs[C]//Proceedings of the 6th workshop on Parallel and distributed systems: testing, analysis, and debugging. ACM, 2008: 6.
[25] Jannesari A, Bao K, Pankratius V, et al. Helgrind+: An efficient dynamic race detector[C]//Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE, 2009: 1-13.
[26] Serebryany K, Iskhodzhanov T. Thread Sanitizer: data race detection in practice[C]//Proceedings of the Workshop on Binary Instrumentation and Applications, 2009: 62-71.
[27] Nethercote N, Seward J. Valgrind: a framework for heavyweight dynamic binary instrumentation[C]//ACM Sigplan notices. ACM, 2007, 42(6): 89-100.
[28] Xie, X. and Xue, J. (2011) Acculock: Accurate and Efficient Detection of Data Races. Proc. 9th Annual IEEE/ACM Int. Symp. Code Generation and Optimization, Chamonix, France, April 6–9, pp. 201–212. IEEE Computer Society, Washington, DC, USA.
[29] Xie, X., Xue, J., and Zhang, J. (2012) Acculock: accurate and efficient detection of data races[J]. John Wiley and Sons Ltd, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom, 2013,43(5): 543–576.
[30] Yu M, Yoo S K, Bae D H. Simple Lock: Fast and Accurate Hybrid Data Race Detector[C]//14th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2013. 2014:50-56.
[31] Yu M, Bae D H. Simple Lock+: Fast and Accurate Hybrid Data Race Detection[J]. The Computer Journal. 2014,59(6): 60-66.
[32] Marino D, Musuvathi M, Narayanasamy S. Lite Race: effective sampling for lightweight data-race detection[C]//ACM Sigplan Notices. ACM, 2009, 44(6): 134-143.
[33] Bond M D, Coons K E, Mc Kinley K S. PACER: proportional detection of data races[J]. ACM Sigplan Notices, 2010, 45(6): 255-268.
[34] Jannesari A, Tichy W F. Identifying ad-hoc synchronization for enhanced race detection[C]//Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on. IEEE, 2010: 1-10.
[35] Jannesari A, Tichy W F. Library-independent data race detection[J]. Parallel and Distributed Systems, IEEE Transactions on, 2014, 25(10): 2606-2616.
[36] Xiong, Weiwei, Park, Soyeon, Zhang, Jiaqi, et al. Ad hoc synchronization considered

harmful[C]// Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2010:163-176.

[37] Sen K. Race directed random testing of concurrent programs[C]//ACM SIGPLAN Notices. ACM, 2008, 43(6): 11-21.

[38] Zhang W, Sun C, Lu S. Con Mem: detecting severe concurrency bugs through an effect-oriented approach[J]. Acm Transactions on Software Engineering & Methodology, 2010, 38(1):179-192.

[39] Wu Z, Lu K, Wang X, et al. Collaborative Technique for Concurrency Bug Detection[J]. International Journal of Parallel Programming, 2015, 43(2):260-285.

[40] Wu Z, Lu K, Wang X, et al. Detecting harmful data races through parallel verification[J]. Journal of Supercomputing, 2015, 71(8):2922-2943.

[41] Lu K, Wu Z, Wang X, et al. Race Checker: Efficient Identification of Harmful Data Races[C]// 2015 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). 2015:78-85.

[42] Kasikci B, Zamfir C, Candea G. Race Mob: Crowdsourced data race detection[C]// Twenty-Fourth ACM Symposium on Operating Systems Principles. 2013:406-422.

[43] Guo Y , Cai Y , Yang Z . AtexRace: across thread and execution sampling for in-house race detection[C]// Joint Meeting on Foundations of Software Engineering. ACM, 2017.

[44] 夏俊鸾, 程浩, 邵赛赛, 等. Spark 大数据处理技术[M]. 北京:电子工业出版社, 2014.

[45] 崔雍浩, 商聪, 陈锶奇, 郝建业. 人工智能综述: AI 的发展[J]. 无线电通信技术, 2019, 45(03): 225-231.

[46] 吴萍, 陈意云, 张健. 多线程程序数据竞争的静态检测[J]. 计算机研究与发展, 2006, 43(2): 329-335.

[47] 魏鹏, 多线程程序中数据竞争的故障动态检测技术研究[D]. 武汉:华中师范大学, 2009.

[48] 姚欣洪, 基于线程摘要的 C/C++数据竞争检测研究[D]. 北京:北京邮电大学, 2011.

[49] 徐超, 基于动态二进制翻译的多线程程序数据竞争检测方法研究[D]. 上海:上海交通大学, 2010.