



## 基于Adaboost模型的并发程序数据竞争语句级检测

孙家泽<sup>1,2</sup>, 易刚<sup>1</sup>, 舒新峰<sup>1,2</sup>

(1. 西安邮电大学 计算机学院, 西安 710121; 2. 西安邮电大学 陕西省网络数据分析与智能处理重点实验室, 西安 710121)

**摘要:** 针对并发程序数据竞争检测时准确率低和开销大的问题, 基于Adaboost模型设计并发程序数据竞争语句级检测方法。对多线程并发程序进行插桩操作, 记录指令的相关内存信息, 并对提取出的指令集做语句级转化处理, 利用语句对相关属性特征构建并发程序Adaboost数据竞争检测模型, 实现多线程程序数据竞争检测工具ADR。实验结果表明, 相比于Eraser、Djit+和Thread Sanitizer工具, ADR能够在降低时间及内存开销的同时, 有效提高分类准确率, 验证了所提方法的有效性。

**关键词:** 数据竞争; 并发程序; 程序插桩; Adaboost模型; 语句级

开放科学(资源服务)标志码(OSID):



中文引用格式: 孙家泽, 易刚, 舒新峰. 基于Adaboost模型的并发程序数据竞争语句级检测[J]. 计算机工程, 2021, 47(12): 215-220.

英文引用格式: SUN J Z, YI G, SHU X F. Data race statement level detection in concurrent programs based on Adaboost model[J]. Computer Engineering, 2021, 47(12): 215-220.

## Data Race Statement Level Detection in Concurrent Programs Based on Adaboost Model

SUN Jiaze<sup>1,2</sup>, YI Gang<sup>1</sup>, SHU Xinfeng<sup>1,2</sup>

(1. School of Computer Science and Technology, Xi'an University of Posts and Telecommunications, Xi'an 710121, China;

2. Shaanxi Key Laboratory of Network Data Analysis and Intelligent Processing, Xi'an University of Posts and Telecommunications, Xi'an 710121, China)

**[Abstract]** To address the low accuracy and high overhead in the data race detection for concurrent programs, a statement-level method for detecting the data race of concurrent programs is designed based on the Adaboost model. The multi-threaded concurrent program is inserted, and the relevant memory information of instructions is recorded. Then the extracted instruction set is transformed at the statement level. On this basis, the model for detecting the data race of concurrent programs is constructed from the relevant attribute characteristics of statements to realize the tool, ADR, for detecting the data race of multi-threaded programs. The experimental results show that compared with Eraser, Djit+ and Thread Sanitizer tools, ADR can effectively improve the classification accuracy while reducing the time and memory overhead. The effectiveness of the proposed method is verified.

**[Key words]** data race; concurrent program; program instrumentation; Adaboost model; statement level

DOI: 10.19678/j.issn.1000-3428.0059734

### 0 概述

随着计算机硬件技术的更新迭代, 多核处理器以及众核处理器<sup>[1]</sup>被广泛应用, 越来越多的开发人员使用并发编程来提高程序的性能, 提升了程序的运行速度、吞吐量和CPU利用率。然而, 并发程序内部存在并发性和不确定性, 这会导致并发程序出现

死锁<sup>[2]</sup>、数据竞争<sup>[3]</sup>、原子性违背<sup>[4]</sup>、顺序违背<sup>[5]</sup>等难以检测和修复的问题。数据竞争往往是引发其他并发问题的根本原因, 在所有的并发缺陷问题中占较大比例, 也是目前研究最多的一类问题<sup>[2]</sup>。如何有效地构建多线程数据竞争检测模型、设计或改进数据竞争定位算法以及准确识别多线程程序中的安全故障, 是多线程并发程序安全性质量保障亟待解决

**基金项目:** 陕西省重点研发计划项目(2020GY-010); 西安市产业研究项目(2019218114GXRC017CG018-GXYD17.10); 西安邮电大学研究生创新基金(CXJLY2019051)。

**作者简介:** 孙家泽(1980—), 男, 教授、博士, 主研方向为群聚智能算法、数据分析; 易刚, 硕士研究生; 舒新峰, 副教授、博士。

**收稿日期:** 2020-10-15 **修回日期:** 2020-12-15 **E-mail:** 770625701@qq.com

的问题,其影响到多核处理器在各个领域的普及与应用。

针对数据竞争检测问题,国内外已有较多研究成果。在动态检测方面,Djit+<sup>[6]</sup>基于发生序关系方法,使用向量时钟对数据竞争进行分析,同时还包括FastTrack<sup>[7]</sup>、LOFT<sup>[8]</sup>和基于锁集的检测工具Eraser<sup>[9]</sup>。在静态检测方面,常用的检测工具有RacerX<sup>[5]</sup>、LOCKSMITH<sup>[10]</sup>和RELAY<sup>[11]</sup>。此外,蔡彦等提出一种基于采样的数据竞争检测方法降低检测的开销,并开发了检测工具AtexRace<sup>[12]</sup>,孙家泽等提出一种基于随机森林的数据竞争指令级检测方法,并实现了AIRaceTest检测工具<sup>[13]</sup>。然而,这些检测方法中仍存在误报、漏报以及开销大等问题。

本文基于数据挖掘分类模型可高效处理大量数据的特点,提出一种以语句对特征作为样本集建立Adaboost<sup>[14]</sup>分类模型的方法,实现多线程程序数据竞争检测工具ADR。通过对被测程序进行动态插桩取样并对插桩<sup>[15]</sup>结果进行语句级转化提取语句对特征,以此构建数据竞争Adaboost检测模型。由于数据竞争最终都可以归结到2条线程交织<sup>[16]</sup>,因此本文分析来自2条不同线程的语句对,以简化检测过程。

## 1 数据竞争语句级检测算法

### 1.1 传统数据竞争检测

在对被测程序进行动态插桩取样时,需要取出指令的所在线程ID、指令的读写操作op和指令所访问的内存地址memaddr。根据数据竞争的定义,如果同时满足以下3个条件,则判定为数据竞争:1)2条指令或多条指令所在线程ID不一样;2)2条指令或多条指令访问的内存地址一致;3)2条指令或多条指令中至少有1条指令的读写操作为写操作。但在实际情况下,该判断方法会带来大量的漏报和误报,因为不同线程存在确定先后关系,或者存在一个公共锁以保证针对共享内存空间的访问不会发生数据冲突,在这种情况下会带来大量的误报。

### 1.2 基于语句对特征的数据竞争检测方法

针对传统数据竞争检测方法误报率高、检测精度低的问题,本文提出一种基于语句对特征的数据竞争检测方法。首先对样本程序集进行插桩,提取出语句对的数据竞争相关属性,然后根据得到的数据集进行模型训练,最后根据得到的分类模型对被测并发程序进行数据竞争检测。本文采用的是集成分类模型。集成分类模型由若干单个分类器组成,相较于单一分类器而言,模型精度会更高。集成分类算法里最常用的是Adaboost算法和随机森林算法,Adaboost相较于随机森林而言,其各个弱分类器

之间存在迭代关系,模型分类准确率更高。因此,本文先采用Adaboost算法建立模型,再利用建立好的模型进行数据竞争检测。

本文方法的具体检测过程如下:

#### 1) 训练过程

(1)设样本并发程序P的指令数为inscount,线程数为 $n$ ,利用插桩工具Pin对被测并发程序P进行动态插桩,记录指令的线程为ID tid,指令访问的内存地址为addr,指令的读写操作为op,指令的锁集为lock。

(2)剔除掉无对应原码的指令以及来自同一条语句的指令,每一条语句仅保留一条指令,将指令级的插桩转换为语句级插桩。

(3)遍历所有的语句对,提取出语句对中的 $P_m$ 、 $P_o$ 、 $P_v$ 、 $P_l$ 、 $P_r$ 这5项数据竞争相关属性。

(4)提取特征 $P_m$ 。语句对 $a-b$ 的访问地址分别为memaddr\_a和memaddr\_b。如果memaddr\_a=memaddr\_b,则 $P_m$ 为1,反之 $P_m$ 为0。

(5)提取特征 $P_o$ 。取出语句 $a$ 对内存读写模式op\_a,取出语句 $b$ 对内存的读写模式op\_b。如果op\_a=op\_b,则 $P_o$ 为1,反之 $P_o$ 为0。

(6)提取特征 $P_v$ 。假设线程总数一定,该值为常量 $M_{\max\text{Threads}}$ ,每一个线程维护一个向量时钟,表示为 $st_i[\cdot]$ ,拥有 $M_{\max\text{Threads}}$ 条记录。所有线程由 $[0, M_{\max\text{Threads}}-1]$ 整数来表示。对于每一个索引 $u$ , $st_i[u]$ 记录当线程 $u$ 与线程 $t$ 同步时线程 $u$ 的本地时间。向量时钟的更新过程如下:先将每一个线程的向量时钟本地时间初始化为1: $\forall t: st_t[t] \leftarrow 1, \forall u: st_u[u] \leftarrow 1$ 。如果线程 $t$ 释放了同步对象,则 $st_t[t] \leftarrow st_t[t] + 1$ ,如果线程 $u$ 获得一个由线程 $t$ 释放的同步对象,则线程 $u$ 更新其向量时钟,且每一个向量的值更新为此时线程 $t$ 和线程 $u$ 的向量的各个维度上的较大值:for  $i = 0$  to  $M_{\max\text{Threads}} - 1: st_u[i] \leftarrow \max(st_u[i], st_t[i])$ 。最后,根据得到向量时钟进行偏序判断,如果两者之间满足偏序关系,则 $P_v$ 为1,反之 $P_v$ 为0。

(7)提取特征 $P_l$ 。对于共享变量 $v$ 和读写锁 $m$ ,每次写访问变量 $v$ 并持有锁 $m$ 时,将锁 $m$ 加入到write\_locks\_held( $t$ )集合中,每次读访问变量 $v$ 并持有锁 $m$ 时,将锁 $m$ 加入到locks\_held( $t$ )集合中。令locks\_held( $t$ )为线程 $t$ 所持有的锁,包括写模式和读模式。令locks\_held( $u$ )为线程 $u$ 所持有的锁,包括写模式和读模式。 $C(v) = \text{locks\_held}(t) \cap \text{locks\_held}(u)$ ,如果 $C(v)$ 为空集,则 $P_l$ 为1,反之 $P_l$ 为0。

(8)提取特征 $P_r$ 。根据样本集已知的结果,如果该语句对发生数据竞争则 $P_r$ 为1,否则为0。

(9)模型训练。以 $P_m$ 、 $P_o$ 、 $P_v$ 、 $P_l$ 为特征属性,以 $P_r$ 为标签属性,训练Adaboost模型。

训练过程的算法描述如下:

输入 样本程序 P1

输出 Adaboost模型

- 1.对样本程序 P1 进行动态插桩
- 2.将指令级插桩转化语句插桩,提取出语句对
- 3.foreach 语句对(a,b) do:
4. $P_m = \text{extract\_Memory}(a,b)$
5. $P_o = \text{extract\_Operation}(a,b)$
6. $P_v = \text{extract\_Vector}(a,b)$
7. $P_l = \text{extract\_Lockset}(a,b)$
8. $P_r = \text{get\_From}(a,b)$
- 9.SampleSet=Combine( $P_m, P_o, P_v, P_l, P_r$ )

2)检测过程

(1)提取被测并发线程  $P_m, P_o, P_v, P_l$  特征属性。

(2)利用训练好的 Adaboost模型对提取得到的属性进行分析,得到检测检测结果  $P_r$ 。

(3)如果  $P_r$  为 0,则发生数据竞争,反之,则没发生数据竞争。

检测过程的算法描述如下:

输入 样本程序 P2

输出 数据竞争检测结果 result

- 1.对样本程序 P2 进行动态插桩
- 2.foreach 语句对(a,b) do:
3. $P_m = \text{extract\_Memory}(a,b)$
4. $P_o = \text{extract\_Operation}(a,b)$
5. $P_v = \text{extract\_Vector}(a,b)$
6. $P_l = \text{extract\_Lockset}(a,b)$
7. $P_r = \text{Adaboost\_Model}(P_m, P_o, P_v, P_l)$
- 8.if  $P_r == 1$ :
- 9.result="发生数据竞争"
- 10.else:
- 11.result="未发生数据竞争"

## 2 Adaboost 模型

本文通过 Intel Pin 工具提取出语句对的特征样本数据,并根据该样本数据建立数据竞争 Adaboost 检测模型。本节具体描述 Adaboost 模型的建立过程。

### 2.1 数据采集

数据采集过程如下:

- 1)输入多线程程序,使用 Pin 工具进行动态插桩取样。
- 2)记录程序中该语句所在的线程 ID tid。
- 3)记录程序中该语句所访问的内存地址 memaddr。
- 4)记录程序中该语句所对应的读写操作 op。
- 5)记录程序中该语句的锁集 lock。
- 6)程序中该语句所在线程的向量时钟 vector-clock。

7)记录程序中该语句所在的行号。

8)输出多线程程序语句的内存信息。

表 1 列出了从多线程程序中随机抽取的 5 条案例语句信息,其中: num 表示当前语句的编号; op 表示当前语句对内存进行的操作,‘R’表示读操作,‘W’表示写操作; memaddr 表示当前语句访问的内存地址; line 表示当前语句在被测程序中的所在行号; tid 表示当前语句所在的线程 ID,其中主线程的 tid 为 0,子线程 thrd1 的 tid 为 1, thrd2 的 tid 为 2, thrd3 的 tid 为 3; locks 表示当前指令收到的锁保护情况,其中 0 表示无任何锁保护; vectorclock 表示当前语句的向量时钟。

表 1 多线程程序语句的内存信息

Table 1 Memory information of multithreading statement

num	memaddr	tid	op	vectorlock	locks	line
1	0x6010a0	3	w	(0,0,0,1)	0	39
2	0x6010a0	3	r	(0,0,0,2)	0	40
3	0x6010a0	2	w	(0,0,1,0)	0	34
4	0x7fdca164a	2	r	(0,0,0,0)	0	35
5	0x6010a0	1	w	(0,1,0,0)	ml	27
6	0x7fdca275c	1	r	(0,1,0,0)	0	29

对表 1 中的数据进行提取,提取出语句对的  $P_m, P_o, P_v, P_l, P_r$  这 5 项特征,如表 2 所示。

表 2 语句对数据数值化特征表

Table 2 Eigenvalue table of statement pair

语句对	$P_m$	$P_o$	$P_v$	$P_l$	$P_r$
39-34	1	1	1	1	1
39-35	0	1	0	1	0
39-27	1	1	1	1	1
39-29	0	1	1	1	0
39-20	1	1	1	1	1
39-21	0	1	1	1	0
40-34	1	1	1	1	1
40-35	0	0	0	1	0
40-27	1	1	1	1	1
40-29	0	0	1	1	0
40-20	1	0	1	1	0
40-21	0	1	1	1	0
34-27	1	1	1	1	1
34-29	0	1	1	1	0
34-20	1	1	1	1	1

### 2.2 Adaboost 模型训练

Adaboost<sup>[14]</sup>算法的设计思想主要是针对同一训练集将其训练成不同的分类器并构建在一起,最终形成一个更强的分类器。Adaboost 算法流程如图 1



所示。首先,每轮迭代中会在新训练集上产生一个新的学习器。然后,使用该学习器对所有样本进行预测,以评估每个样本的重要性。在每一次迭代时,会给每一个样本赋予一个权重,根据每一次预测的准确性进行动态调整,权重越高的样本会在下一次训练过程中占越大的比重,即准确性低的样本会被着重关注。

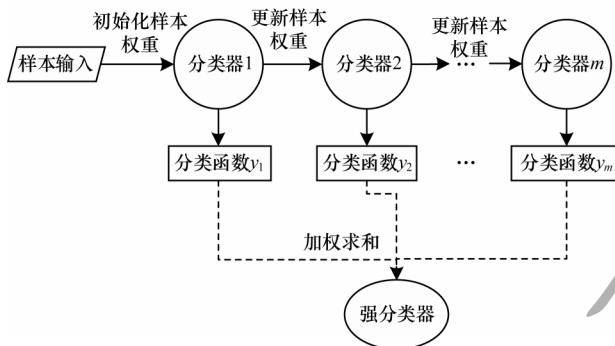


图1 Adaboost 算法流程

Fig.1 Procedure of Adaboost algorithm

本文从 Github<sup>[17]</sup>上选取 5 组多线程程序来训练 Adaboost 模型,取出的语句对经整合后总数为 60 8631。对整理得到的数据进行预处理得到特征数据集 TestUnity。在特征数据集 TestUnity 上进行 Adaboost 模型训练时,如果迭代次数  $n\_estimators$  设定得较低,必然导致模型精确度降低,但如果  $n\_estimators$  设定得较高,也会导致训练模型的开销增大。因此,需要对其设定一个合适的值,使模型的精确度达到要求并且开销最低。同时,如果决策树最大深度  $max\_depth$  过大也会出现拟合现象,影响模型精确度。

图2显示了不同的决策树深度随着迭代次数的增加,其分类误差率的变化趋势,从中可以看出:当  $max\_depth=10$  时,随着迭代次数的增加,模型的分类误差率无明显变换,即泛化能力没有增强,模型处于过拟合状态;当  $max\_depth=1$  时,迭代次数逐渐增加时,分类的误差率也逐渐减少,同时其泛化能力也逐渐增强;但是当  $n\_estimators \geq 90$  时,模型的分类误差率趋于平稳,而当  $max\_depth=2$  时,其误差率走势与  $max\_depth=1$  时相似,但其分类误差率相对更低。由于在 Adaboost 模型中选定的弱学习器的复杂度很低,其决策深度一般设定在 1 到 2 之间,因此设定最优的  $max\_depth=2$ ,而当  $n\_estimators \geq 90$  时,由于模型的分类误差率趋于平稳,随着迭代次数继续增加必然导致模型训练的开销也会不断增加,因此,本文设定最优的  $n\_estimators$  为 90。

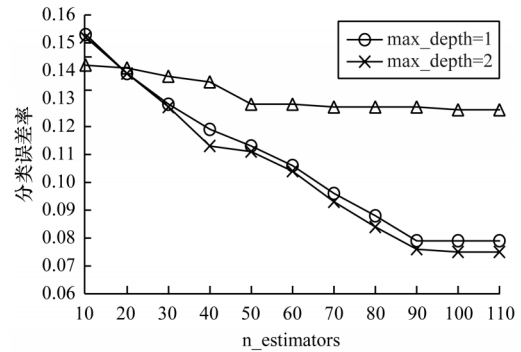


图2 分类误差率与迭代次数的关系

Fig.2 Relationship between classification error rate and iteration times

### 3 实验与结果分析

本节通过一系列实验,验证所提方法的有效性。首先提出研究问题,然后介绍实验数据集,最后总结并分析实验结果。

#### 3.1 研究问题

针对本文所提出的多线程程序数据竞争检测工具 ADR,从以下 2 个角度出发进行有效性验证。

1) 与 Eraser、Djit+ 等数据竞争检测工具相比,ADR 检测准确率是否提升以及提升幅度。

2) 与 Eraser、Djit+ 等数据竞争检测工具相比,ADR 的检测开销是否有所降低。

#### 3.2 实验数据集

由于目前没有任何机构或者研究人员发布数据竞争检测的精确数据集,因此本文选择 Github<sup>[17]</sup>上的 1 组已知数据竞争结果的多线程并发程序进行插桩取样,将插桩结果作为建立 Adaboost 模型的训练集。

实验环境为 Ubuntu19.04,处理器核心数为 4,运行内存 4 GB。选择的验证测试程序来源如下:

1) 为验证 Adaboost 数据竞争检测模型的准确性,在 Google Code 的 data-race-test<sup>[18]</sup>测试集中筛选出 50 组多线程并发程序组成一个测试程序 TestCodes 进行验证。

2) 为验证 Adaboost 数据竞争检测模型的时间开销和内存开销,选择 SPLASH-2 基本套件<sup>[19]</sup>中的 2 组程序和 Parsec 基准程序 3.1<sup>[20]</sup>中的 3 组程序来进行验证。具体如表 3 所示。

表 3 测试程序信息

Table 3 Information of test programs

测试程序	线程数	来源
radix	4	SPLASH-2 基本套件
fft	8	SPLASH-2 基本套件
streamcluster	17	Parsec 基准程序 3.1
deup	25	Parsec 基准程序 3.1
x264	64	Parsec 基准程序 3.1

### 3.3 实验过程

ADR 体系结构如图 3 所示, 主要由模型构建模块和模型预测模块这 2 部分组成。本文首先利用样本程序的插桩结果训练得到 Adaboost 分类模型, 然后利用建立好的分类器对待测程序进行数据竞争检测。

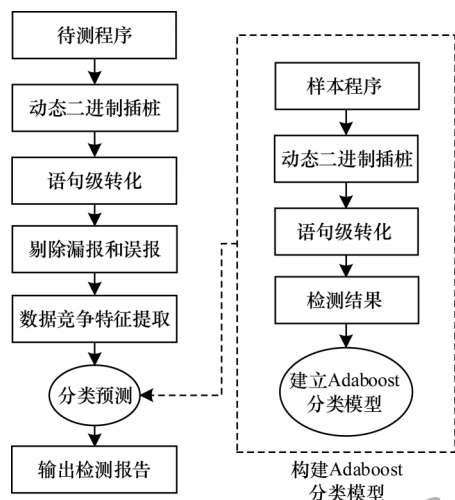


图 3 ADR 体系结构

Fig.3 Architecture of ADR

将 ADR 与 Eraser、Djit+ 和 Thread Sanitizer 这 3 种动态数据竞争检测工具进行对比实验。Eraser、Djit+ 和 Thread Sanitizer 分别使用的检测算法是 Happen-Before、Lockset 和 Happen-Before+Lockset 混合算法。

### 3.4 检测精度分析

由于 TestCodes 程序由多个小程序示例组成, 因此每个小程序示例可能存在零个或者多个数据竞争。本文定义以下 3 个指标来衡量检测精度: 数据竞争检测的误报数 FP, 数据竞争检测的正确数 TP, 数据竞争检测的漏报数 FN。4 种检测工具对 TestCodes 程序的数据竞争检测结果如图 4 所示, 已知 TestCodes 中真实存在的数据竞争次数为 80。

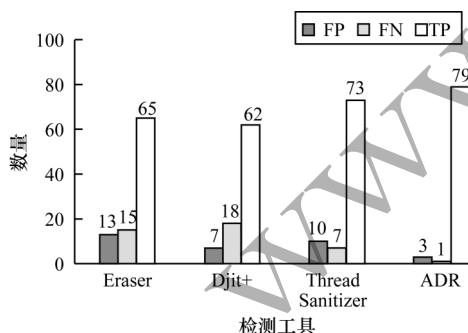


图 4 不同数据竞争检测工具的检测精度

Fig.4 Detection accuracy of different detection tools

从 TP 值上来对比分析。ADR 检测到 79 次数据竞争, 其检测准确度最高。Djit+ 容易受到线程调度的影响, 所以漏报较多, Eraser 与 Thread Sanitizer 由于是在原码级别进行的判断, 会遗漏部分数据竞争。ADR 采用的 Adaboost 分类模型是由若干个弱分类器组合而

成, 而且各个弱分类器之间存在迭代关系, 使模型分类准确率更高。

从 FP 值上来对比分析。因为 Eraser 对很多确定性同步语句未进行处理, 所以其误报数最多。Djit+ 由于其使用的算法是 Happen-Before, 而 ADR 是在语句级上进行的实验, 剔除了重复以及无对应原码的指令信息, 因此降低了检测的误报数。

### 3.5 检测开销分析

本文选取 SPLASH-2 基本套件中的 radix、fft 程序和 Parsec 基准程序 3.1 中的 streamcluster、deup、x264 组成一个测试程序集, 它们的线程数量呈递增模式, 分别为 5、11、17、25、64, 对 Eraser、Djit+、Thread Sanitizer 与 ADR 检测工具进行时间开销与内存开销方面的对比, 如图 5 和图 6 所示。

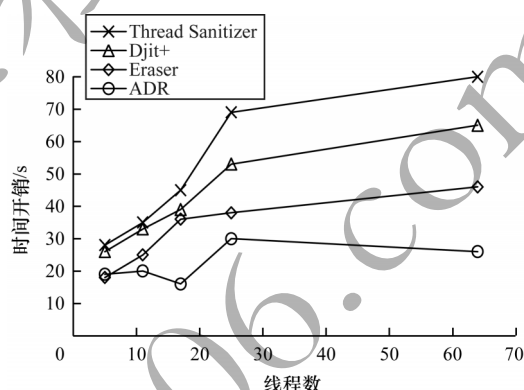


图 5 不同数据竞争检测工具的检测时间开销

Fig.5 Comparison of detection runtime overhead of different detection tools

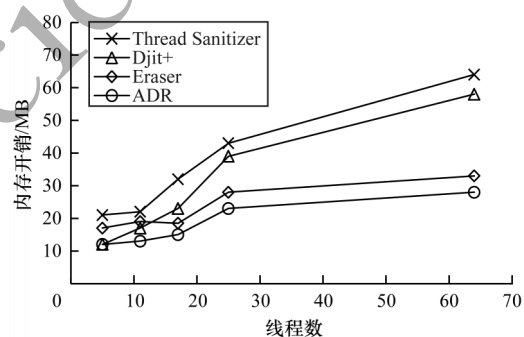


图 6 不同数据竞争检测工具的检测内存开销

Fig.6 Comparison of detection memory overhead of different detection tools

由图 5 可以看出, Thread Sanitizer 的检测时间开销最大, Djit+ 的时间开销仅次于 Thread Sanitizer。这是因为 Thread Sanitizer 和 Djit+ 使用的都是 Happen-Before 算法, 当线程数量很多时, 在各个线程之间状态切换非常麻烦, 增加了时间的开销, 且 Thread Sanitizer 还需要判断锁集的状态, 所以时间开销最大。ADR 通过高效处理大量数据建立检测模型加快了检测的速度, 使检测的时间开销更小。综合检测数据可知, ADR 检测的时间开销相比于其他检测方法降低约 29.3%。

由图6可以看出, Thread Sanitizer相比于其他的检测方法, 其检测的内存开销最大。这是因为 Thread Sanitizer 保存了所有并发历史访问的锁集和向量时钟, 而 Djit+ 保留了所有的历史访问的向量时钟, 因此, Thread Sanitizer 产生的内存开销最大。ADR 为语句级检测, 在对待测程序进行插桩取样时, 每一条语句仅保留一条指令, 只需要记录各个线程在内存中的状态信息, 剔除掉了无对应原码的指令, 降低了内存开销。综合检测数据可知, ADR 检测所产生的内存开销相比于其他检测方法降低约 21.1%。

#### 4 结束语

本文提出一种基于语句对特征的数据竞争方法。利用插桩工具 Pin 对被测并发程序进行动态插桩, 记录指令的相关内存信息。在此基础上, 通过对指令信息进行过滤操作, 将指令级插桩转换成语句级插桩, 然后根据样本数据集建立数据竞争 Adaboost 模型, 对被测程序进行数据竞争检测, 并基于此模型实现数据竞争检测工具 ADR。实验结果表明, 该方法能够有效降低漏报率与误报率, 同时检测过程不受线程调度的影响, 可减少检测的时间开销和内存开销, 提高检测效率。后续将在数据采集过程中引入采样策略, 减少样本的数据量, 从而进一步减少数据竞争检测的内存消耗, 优化数据竞争的检测效率。此外, 在基于语句对特征的数据竞争方法中融入更优的采样策略, 也将是下一步的研究方向。

#### 参考文献

- [1] 朱怡安, 黄林林, 李联, 等. 多核平台下分区操作系统的安全关键任务调度方法[J]. 计算机工程, 2017, 43(12): 38-44.  
ZHU Y A, HUANG L L, LI L, et al. Safety-critical task scheduling method for partitioned operating system in multi-core platform[J]. Computer Engineering, 2017, 43(12): 38-44. (in Chinese)
- [2] 黄理, 顾乃杰, 曹华雄. 基于 Petri 网的多线程程序死锁检测[J]. 计算机工程, 2016, 42(4): 1-6.  
HUANG L, GU N J, CAO H X. Deadlock detection in multi-threaded program based on Petri net[J]. Computer Engineering, 2016, 42(4): 1-6. (in Chinese)
- [3] NETZER R H B, MILLER B P. What are race conditions? Some issues and formalizations [J]. ACM Letters on Programming Languages and Systems, 1992, 1(1): 74-88.
- [4] 胡敏, 陈雨亭. 基于距离挖掘的多变量原子性违例检测[J]. 计算机工程, 2012, 38(13): 61-63, 74.  
HU M, CHEN Y T. Multi-variable atomicity violation detection based on distance mining [J]. Computer Engineering, 2012, 38(13): 61-63, 74. (in Chinese)
- [5] ENGLER D, ASHCRAFT K. RacerX: effective, static detection of race conditions and deadlocks[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 237-252.
- [6] 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.
- [7] FLANAGAN C, FREUNDS N. FastTrack: efficient and precise dynamic race detection[C]//Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, USA: ACM Press, 2009: 121-133.
- [8] CAI Y, CHAN W K. LOFT: redundant synchronization event removal for data race detection[C]//Proceedings of the 22nd International Symposium on Software Reliability Engineering. Washington D. C., USA: IEEE Press, 2011: 160-169.
- [9] 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, 31(5): 27-37.
- [10] PRATIKAKIS P, FOSTER J S, HICKS M. Locksmith: context-sensitive correlation analysis for race detection[J]. ACM SIGPLAN Notices, 2006, 41(6): 320-331.
- [11] 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 Symposium on the Foundations of Software Engineering. New York, USA: ACM Press, 2007: 205-214.
- [12] GUO Y, CAI Y, YANG Z J. AteRace: across thread and execution sampling for in-house race detection [C]//Proceedings of Joint Meeting on Foundations of Software Engineering. New York, USA: ACM Press, 2017: 1-5.
- [13] 孙家泽, 阳伽伟, 杨子江. 多线程程序数据竞争随机森林指令级检测模型[J]. 清华大学学报(自然科学版), 2020, 60(10): 804-813.  
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. (in Chinese)
- [14] 曹莹, 苗启广, 刘家辰, 等. AdaBoost 算法研究进展与展望[J]. 自动化学报, 2013, 39(6): 745-758.  
CAO Y, MIAO Q G, LIU J C. Research progress and prospect of AdaBoost algorithm [J]. Acta Automatica Sinica, 2013, 39(6): 745-758. (in Chinese)
- [15] SCHARDL T B, DENNISTON T, DOUCET D, et al. The CSI framework for compiler-inserted program instrumentation[J]. Performance Evaluation Review, 2019, 46(1): 100-102.
- [16] 操旺根. 并发程序数据竞争检测方法研究和分析[J]. 信息技术与信息化, 2019(12): 171-173.  
CAO W G. Research and analysis of concurrent program data competition detection method [J]. Information Technology and Informatization, 2019(12): 171-173. (in Chinese)
- [17] Data race benchmark [EB/OL]. [2020-10-10]. <https://github.com/arnabd88>.
- [18] Data-race-test [EB/OL]. [2020-10-10]. <https://code.google.com/p/data-race-test/>.
- [19] FARCHI E, NIR Y, UR S. Concurrent bug patterns and how to test them [C]//Proceedings of 2003 International Symposium on Parallel and Distributed Processing. Washington. D. C., USA: IEEE Press, 2003: 286-296.
- [20] BIENIA C. Benchmarking modern multiprocessors [D]. Princeton, USA: Princeton University, 2011.