DataCollider 调研报告

刘伟森 PB15111595
王泽凡 PB15111593

一、 概述

DataCollider 是一个动态检测内核模块中数据竞争的轻量级工具。它能够绕开传统的程序分析方法,利用现有硬件结构中的调试寄存器对复杂的内核代码进行有效的数据竞争检测,并通过采样少量的访存操作来降低系统的运行时开销。

二、 背景

2.1 数据竞争的定义

两个访问内存的操作有冲突，如果：

（1） 访问的物理地址相交
（2） 至少有一个是写操作
（3） 没有正确同步

如果同时执行两个冲突的内存访问（由处理器或任何其他设备执行），那么程序会产生数据竞争。

2.2 数据竞争检测相关工作

数据竞争检测可以大致分为静态竞争检测和动态竞争检测。静态竞争检测通常在不直接执行程序的情况下分析源代码或字节代码，动态竞争检测则运行程序并监控其执行情况。

Happens-before 方法的基本原理是通过推断多线程程序中可能的指令发生顺序来报告潜在的竞争性线程交错顺序。对于任意两个冲突的访问 A 和 B，我们只需检查 A 是否发生在 B 之前，或者 B 发生在 A 之前，或者没有先后关系。若两个冲突的指令间不存在 happens-before 关系，就可以推断它们构成了一个数据竞争。尽管 happens-before 方法不会产生误报，但大量的记录和推断工作仍然是非常复杂和耗时的。

lock-set 方法是在每个线程访问共享数据时检查它所拥有的锁的集合，计算该处内存地址所有锁集合的交集。若该交集为空，检测工具则会报告此共享数据未被保护且有可能发生数据竞争。lock-set 方法的主要限制是它不检查真正的数据竞争，而是是否违反特定的锁规则。不幸的是，许多并发程序（特别是内核代码）使用复杂的锁规则，并且还使用锁以外的同步机制。

2.3 相关工作的不足之处

并发程序中的数据竞争问题是一类很重要的并发错误,数据竞争的影响可能包括程序崩溃、更新丢失以及难以重现和调试的数据损坏，但是数据竞争难以被检测和修复。以往的研究大多针对用户层的数据竞争检测，并在此问题上取得了重大的进展,但在操作系统内核层面的数据竞争问题却几乎没有涉及。内核代码使用的同步机制远比用户层应用程序中复杂,如不同种类的锁、软硬件中断、大量的信号量原语以及各种底层的共享资源等。这些差别使得原有的用户层检测方法很难被应用到内核环境中。

三、 实现

DataCollider 利用当前计算机硬件系统结构中的代码断点和数据断点，通过采样程序中的少部分访存操作进行数据竞争检测。因此它不会给未被采样的代码区域带来额外的运行时开销，只需设置一个低采样率就可达到以微小开销开展数据竞争检测工作的目的。

首先，对内存访问指令进行采样，在选中的位置设置代码断点，当执行到断点时，DataCollider 会暂停当前正在执行的线程，产生延迟，然后检测其它线程是否有冲突的访问。DataCollider 使用两种冲突检测策略：数据断点和重复读取。基本算法如下：

```
AtPeriodicIntervals() {
  // determine k based on desired
  // memory access sampling rate
  repeat k times {
    pc = RandomlyChosenMemoryAccess();
    SetCodeBreakpoint( pc );
  }
}

OnCodeBreakpoint( pc ) {
  // disassemble the instruction at pc
  (loc, size, isWrite) = disasm( pc );
  DetectConflicts(loc, size, isWrite);
  // set another code break point
  pc = RandomlyChosenMemoryAccess();
  SetCodeBreakpoint( pc );
}

DetectConflicts( loc, size, isWrite) {
  temp = read( loc, size );
  if ( isWrite )
    SetDataBreakpointRW( loc, size );
  else
    SetDataBreakpointW( loc, size );
  delay();
  ClearDataBreakpoint( loc, size );
  temp' = read( loc, size );
  if( temp != temp' || data breakpoint fired )
    ReportDataRace( );
}
```

3.1 采样算法

设计一个用于数据竞争检测的良好采样算法存在几个挑战。首先，数据竞赛涉及两个内存访问，这两个访问都需要被采样到。如果内存访问是独立采样的，那么发现数据竞争的概率是个体抽样概率的乘积。DataCollider 通过对第一个访问进行采样并使用数据断点捕获第二个访问来避免这种乘法效应。这使得 DataCollider 在低采样率下依然有效。

其次，数据竞争是小概率事件，大多数访存指令不会导致数据竞争。采样算法的思想是，如果程序中某处是有问题的如访问共享数据时未正确使用同步机制，那么该问题程序的每一次动态执行都有可能导致数据竞争。因此，DataCollider 进行静态采样而不是动态采样。静态采样器对执行次数少的指令和频繁执行的指令提供同等的优先权。

给定一个并行程序的二进制文件，DataCollider 反汇编这个二进制文件，生成一个包含所有访问内存的代码位置的采样集。DataCollider 运用一个简单的静态分析方法来从采样集合中排除那些只访问了线程本地堆栈地址的指令。同样地，它还会排除一些访问了标记为“volatile”的内存地址或使用了硬件同步原语的指令。

DataCollider 通过插入代码断点来从采样集中对程序位置进行采样。初始断点设置在从采样集合中随机选择的少量程序位置处。如果当代码断点触发时，DataCollider 将在该断点处对内存访问执行冲突检测。然后，DataCollider 从采样集合中随机选择另一个程序位置，并在该位置设置一个断点。

该算法对采样集中的所有程序位置进行统一采样，而与这些代码的执行频率无关。这是因为插入代码断点的选择是针对采样集合中的所有位置随机进行的。经过一段时间的运行，设置断点的位置就可能选择在一些很少被执行到的程序地址上，提高了这些地方获得数据竞争检测机会的可能性。

3.2 检测策略

对于采样出来的内存访问，DataCollider 会暂停当前线程，产生一个延迟，等待看是否有另一个线程对相同内存地址进行冲突访问。它使用两种策略：数据断点和重复读取。DataCollider 同时使用这两种策略，因为每种策略都补充了另一种策略的弱点。

3.2.1 数据断点

现有的硬件体系结构提供了在处理器读取或写入某个内存地址时进行捕获的功能，这对于有效地支持在调试器中设置数据断点起到非常关键的作用。DataCollider 利用了 x86 硬件提供的 4 个数据断点寄存器对可能与被采样的访存发生冲突的其他访存操作进行了有效的监控。

若当前访存是写操作，DataCollider 会令处理器处于等待捕获对该地址的读操作或写操作的状态。若是个读操作，就只令处理器等待捕获对该地址的写操作即可，因为对同一地址的两个读操作不会发生数据竞争冲突。经过一定的延迟之后，如果没有检测到冲突的操作，就会清除数据断点寄存器。当数据断点触发时，DataCollider 就成功检测到一个数据竞争。更重要的是，它是在发生现场捕获的，此时两个线程正在对同一个内存位置进行冲突访问。

数据断点是根据虚拟地址进行检测的，但是两个并发线程访问相同的虚拟地址但不同的物理地址时不会产生竞争。在 Windows 中，大多数内核驻留在相同的地址空间中，但有两个例外。

如果线程在不同进程中执行，则访问用户地址空间的内核线程不会发生冲突。如果采样访问位于用户地址空间中，则 DataCollider 不使用断点，并且默认使用重复读取策略。

内核地址空间称为会话内存，相同的内核地址可以根据进程所属的会话映射到不同物理地址。当采样访问位于会话内存空间中时，DataCollider 会设置数据断点，但会在向用户报告冲突之前检查冲突访问是否属于同一个会话。

3.2.2 重复读取

重复读取策略依赖于一个简单的思想：如果冲突的写入更改了内存位置的值，则 DataCollider 可以通过重复读取内存位置检查以检查值更改来检测此情况。在程序执行到

断点处时，先读取一次该内存的值，经过一定的延迟后，再次读取该内存的值。如果两个值不同，那么就发生了数据竞争。

这种方法的一个明显的缺点是它无法检测到冲突读取。同样，它不能检测到多个冲突写入，其中最后一个写入与初始值相同的值。重复读取策略仅捕获两个线程中的一个数据竞争现场，这使得调试数据竞争变得更加困难，因为我们不知道哪个线程或设备进行冲突写入。

3.3 延迟控制

对于采样的内存访问，DataCollider 会尝试通过延迟线程一段时间来检测对同一内存位置的冲突访问。为了 DataCollider 成功，这个延迟必须足够长，以便发生冲突访问。 另一方面，延长线程时间过长可能是危险的，特别是如果线程拥有一些对整个系统正常运行至关重要的资源。

根据执行线程的 IRQL（Interrupt Request Level），DataCollider 将线程延迟预设为对应级别的最长时间。级别越高，延迟时间越短。

在高于 DISPATCH 级别（内核调度程序的级别）的 IRQL 上，DataCollider 不会插入任何延迟。虽然在这个级别可以插入一个小窗口来确定中断服务程序之间可能的数据竞争，但是由于延迟过短，DataCollider 很难检测到。

在 DISPATCH 级别运行的线程不能让处理器切换到另一个线程。因此，延迟只是一个忙等，目前这个级别的线程的延迟是少于 1 毫秒的随机时间。对于较低的 IRQL，DataCollider 通过轮询来产生延迟，最多延迟 15 ms。在轮询中，线程通过检查断点触发率来查看其他线程是否正常运行。如果未检测到进度，该线程会提前停止等待。