《Scalable and Incremental Software Bug Detection》Review

论文信息

《Scalable and Incremental Software Bug Detection》 Scott McPeak, Charles-Henri Gros, Murali Krishna Ramanathan

内容概述

本文提出了一种分布式的bug检测框架，在保证检测过程迅速、检测结果确定的前提下，引入了增量分析的机制，减少了分析过程中不必要的重复分析，大大降低了检测的时间开销。

技术难点

Work-unit的设计

保证并行性，单元之间的相对独立性；当分析对象出现少量改动时，受影响的work-unit要尽可能小，比如在设计work-unit时不能将代码行编号显式地包含在work-unit中，当在源文件开头加一个空行时，后面的每行代码的行编号发生改变，此时每个work-unit需要重新分析，然而代码的语义并未发生改变。

调度结构设计

为了保证整体分析的分布式特性，架构中包含Analysis Master、Analysis Worker、Distributor Master、Distributor Worker，并设计了三个队列，分别存储待处理的workunit、Worker中的workunit（最多两个）、返回的workunit结果。这一机制的引入保证了整个分布式调度的效率。

过程间分析

完整的过程间分析框架采用的是传统的抽象解释的流程，能够检测堆溢出、空指针异常等语言缺陷。在这篇论文的工作中，采用了多次分析：先自下而上，然后自上而下，最后自下而上进行分析。每一个函数由一个处理节点进行处理，这样的隔离处理机制保证了增量式分析能够得以实施。

其他技术

文中还引入了其他的优化策略，比如在调度时优先处理deriver的部分，因为checker的部分能够以很小的时间代价进行计算。

影响并行化的因素

函数调用图的结构，有的程序难以进行并行分析； Analysis Master部分是瓶颈，复杂的类层次关系导致work unit的构造过程复杂。

可以改进的部分

1. 本文对框架的部分介绍很详细，但是对于具体的缺陷检测部分没有详细说明。
2. 在Distributor Worker的部分，一个Distributor Master可以调用n个worker，但是如何选择n个worker会影响到整个分析的效率；文中仅仅说明了n由实际的硬件条件与用户的偏好进行制定，但是更好的解决策略是根据实际的任务执行情况进行实时调整。 比如假设Q1中的work units的个数为k，可以设置n为k的单调增函数。当k增加时，也相应地增加Distributor Workers的数目，这样地动态调整能够保证Q1中的work unit能无阻塞地被Distributor Workers处理，提高整体任务处理速度。
3. 在图3的过程间分析的整体架构中函数的AST树作为分析的输入，这里文中的工作没有利用函数之间的调用关系。比如某些函数被反复调用，这些函数在项目中的重要性相对更高，因此可以进行多入口分析，优先分析重要度程度高的函数。另一方面，可以根据函数的调用关系图将所有待分析的函数划分成若干个连通支，同一个连通分支交付给一个线程进行分析，由此可以实现更高的并行效果。
4. 为了实现更有效地增量式的分析，3中提到的连通分支的构建就显得更加必要和有效。当函数f的内容发生变化时，它只会影响到与f在同一个连通分支的函数的分析，其他连通分支内的函数不需要重复分析。因此可以动态地生成函数的调用关系图，这一过程的资源开销较小，附加的操作不会影响整个工具的时间开销。