读博士的难与易

蒋炎岩
南京大学

关键词：并发程序动态分析　读博感悟

为了实现并发程序的动态分析 (dynamic analysis)，必须在运行时对并发程序进行观测。我的博士学位论文揭示了观测并发程序执行的固有困难和在此意义下所做的一些尝试。“困难”和“容易”也一直交替出现在我攻读博士学位的这些年中，故把其中值得借鉴的经历与年轻的博士生们分享与共勉。

观测并发程序执行的难与易

从“哲学家吃饭”问题说起

“哲学家吃饭问题”是大学时代学习并发编程时的经典难题。今天，哲学家们来到舌尖上的中国，面对一桌丰盛的菜肴，他们不再受限于必须同时拿起手边的两根筷子才能用餐，因此可以更尽兴地享用美食。另一方面，吃中餐的哲学家们也带来了一个新的问题——聚餐是一个很复杂的过程，我们能否把吃饭时发生的事情像讲故事一样，完整地复述出来？

如果把哲学家看作是线程，每道菜看作是变量，那么每个哲学家在吃饭的过程中都可以多次执行以下两个操作：(1) 观察某一道菜，在瞬间记下这道菜的模样（相当于读出变量的值）；(2) 吃某一道菜，在瞬间改变它的模样（相当于为某个变量写入一个值）。哲学家们都是了不起的天才，在吃饭的过程中都已默默地记下自己所知的一切，即每个线程记录下自身发生的读 / 写操作的变量和它们的数值。

之后的一天，哲学家们重新聚在一起，试图解决“哲学家吃中餐问题”：他们各自整理了对那一天吃饭过程的记忆，并试图一起还原出吃饭的完整过程。更确切地说，我们希望在程序执行结束后，把线程本地的读 / 写操作日志合并成全局的变量读 / 写序列，相当于给所有读 / 写事件分配发生的先后次序，使得每个读事件读出的数值都等于最近一次对该变量写入事件写入的数值（即根据线程本地日志得到全局满足顺序一致性的事件排序），如图 1 所示。吉本斯 (Gibbons) 和科拉赫 (Korach) 在 1997 年对于“哲学家吃中餐问题”给出了一个颇为悲观的答案 [1]：如果 P ≠ NP，再聪明的哲学家也无法在多项式时间里恢复出满足顺序一致性的事件排序。

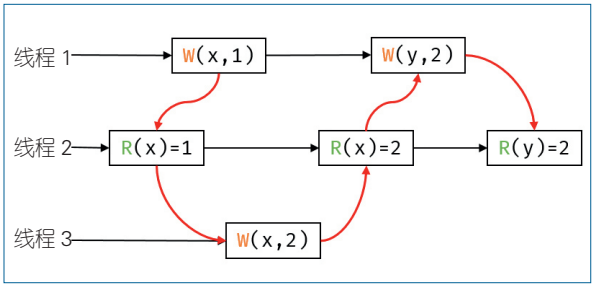


图 1　“哲学家吃中餐问题”示意图。R/W 分别代表读 / 写事件，红色表示满足顺序一致性的线程调度

观测并发程序的执行

“哲学家吃中餐问题”为什么重要？想想我们身边的并发程序——操作系统内核、云计算 / 分布式系统、服务器应用，大家都有的体验是它们很难写、更难写对。因为不确定性的存在，即使我们已经知道程序里有缺陷 (bug)，要想复现缺陷触发的过

程都很困难，更不用说测试了。哲学家吃中餐问题恰恰就是要解决从日志中恢复出程序执行过程的难题，这对并发程序调试技术的重要性是不言而喻的。不仅是调试，我们还可以针对观测到的一次执行预测程序中的缺陷，例如数据竞争或 use-after-free；甚至在运行时就对程序的行为进行干预，避免并发缺陷被触发。这一大类基于程序执行轨迹实现的技术被统一称为并发程序的动态分析技术。

“哲学家吃中餐问题”的 NP- 完全性（固有的困难）并不意味着我们无法在实际中解决它。为了实现并发程序的动态分析，必须在运行时观测并发程序的执行，而观测却并不局限于“线程本地的记录”。我们希望有一台“摄像机”拍摄下哲学家吃饭的全过程，而修改程序的代码或运行时环境，恰好能实现这样的摄像机：为所有共享内存的读 / 写操作上锁，并在锁的保护下记录它们发生的顺序，这样就得到了共享内存访问的日志，如图 2 所示。从此意义上看，观测并发程序执行又是容易的。

```
► spin_lock(log->lock);
data<-load(addr);
► log->append(gettid(), addr, data);
► spin_unlock(log->lock);
```

图 2　修改程序代码对共享内存读操作进行观测（标记“►”的为插入的代码）

难与易之间的矛盾是观测并发程序执行的开销。虽然锁可以实现有效的观测，但却降低了系统的并行度、拖慢了程序的执行。并发性存在于计算机系统栈的各个层次上，因此来自体系结构、计算机系统、程序设计语言和软件工程领域的研究者，全都对实现高效的并发程序动态分析有兴趣。在 30 多年的研究历程中，我们看到了很多非常精彩的锁机制的实现：借助硬件的缓存一致性协议 [2]，使用虚拟机、分支计数和用分页机制实现的 CREW 协议 [3, 4]，以及在线程局部访问下“乐观”的锁机制 [5]。另一方面，我们可以通过记录间接信息（如每个线程执行的路径），使用约束求解的方式去恢复共享内存访问的日志 [6]。我的博士论文在运行时获取共享内存访问数据依赖这一问题上做了一些微小的贡献，主要集中在利用局部性减少锁的开销。例如，我们发现在空间（地址）上连续的多个变量，例如数组的一部分，在共享内存访问的意义上通常可以看成是一个大的“虚拟变量”。如果以虚拟变量为单位进行记录，则可以有效减少锁的数量，并将在线日志的大小减少到最多三个数量级 [7]。

在观测并发程序执行的基础上就可以实现各式各样的动态分析技术了。这里举一个我们测试并发程序的例子 [8]：试想我们把线程中的读 / 写事件按顺序串在一根绳子上（图 3），处理器按照固定的速度按顺序执行绳子上的事件。改变绳子的长度（例如拉绳子相当于把线程执行的速度变慢）就得到了不同的线程调度——那些触发并发缺陷的调度，很可能隐藏在某些绳子长度的配置中。我们设计了聪明的策略生成具有多样性的线程调度，在先前研究者已经反复测试过的并发程序上找到了前所未知的并发缺陷。对于复杂性日渐增长的并发程序来说，动态分析是一项非常有前景的技术，对具体内容感兴趣的读者可以参考我们的中文综述 [9]。

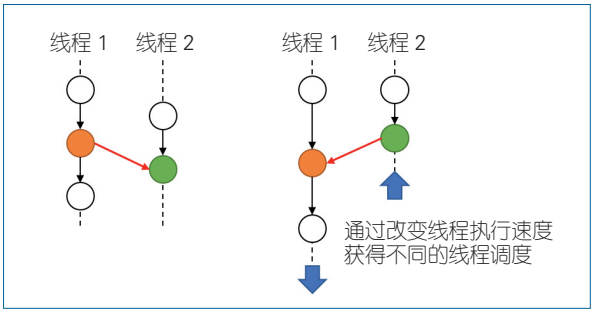


图 3　基于线程调速的并发程序测试技术示意图

有没有免费的午餐？

观测并发程序执行是非常基础且重要的问题，来自各个领域的研究者都取得了丰硕的成果，但唯独“完美”现在还做不到。现有观测并发程序执行的工作可分为两类：要么存在一个 NP- 难的最坏情况，要么不可避免会在某些情况下用锁（类似图 2 的方式）保证一次观测不被其他线程打断。

“哲学家吃中餐问题”的 NP- 完全性一定程度上反映了现有研究工作面临的困境——如果只允许程序进行少量的线程本地记录，则恢复满足顺序一致性的全局调度是困难的。另一方面，我们也已经知道共享内存上的互斥和可序列化并发对象必须借助读 - 写原子操作才能实现 [10]。这启发我们提出了一个猜想：观测并发程序的执行没有免费的午餐 (no-free-lunch)。具体的陈述是，在一个限定的计算模型下，即便允许对并发程序进行一定程度的修改（在共享内存访问前后插入一定数量的共享内存读 / 写操作，其中写操作只限于为观测并发程序执行而额外分配的内存），只要这些插入的读 / 写序列是等待无关 (wait-free) 的，就存在某种线程调度，根据线程本地的观测结果恢复满足顺序一致性的程序执行是 NP- 完全的。

如果这个猜想成立，将对这 30 年的研究成果给出一个“二分性”的总结——想要观测并发程序的执行，要么添加处理器之间的同步，要么付出 NP- 完全的代价。这意味着观测并发程序执行是既困难又容易的。目前我们相信这个猜想是成立的。在试图证明它的过程中，我们对“哲学家吃中餐问题”给出了的一个新的（简化的）证明，并据此得出了一些有用的结论，例如在对程序的修改能写成只读前缀 + 只写后缀的情形下的 NP- 完全性。受限于掌握的数学工具，我们还没能完全证明或否定这个猜想。如果猜想被推翻，我们就更惊奇了——说明 30 年来大家的努力都有本质上的不足，抑或 P = NP，也许我们就需要重新理解整个计算机科学了。

一方面，观测并发程序的执行在理论上有无法逾越的复杂性界限；另一方面，我们又有一些非常简单却在实际中行之有效的观测方法。在此之中，“没有免费午餐”的猜想本身已经带给我们很多启迪和畅想——它必须在很多限定条件下才能成立。因此只要假设的条件被推翻，观测并发程序执行就变得不困难了。例如，我们的复杂性结论是在“最坏情况”下得出的，即存在一个 NP- 完全的“极端”线程调度。但最坏情况也许像平滑分析 [11] 中指出的那样，在实际中很难存在。现实中的并发程序执行有它的特征（例如各种局部性），也被我们用来降低观测并发程序执行的开销。就在难与易之间，理论和实践得到相互的印证。然而在理论与实践中，我们都仍有很多未解决的难题，博士读了很多年，反而觉得研究才刚刚开始，不像是告一段落。

读博士的难与易

面对难题

网络上大家常说“读博是个很讲运气的事”，碰上好的导师、好的方向，博士就读得顺风顺水，否则连毕业都是遥远的奢望。我读博期间遇到的第一个难题就与此相关：我遇到了一位好的导师，但不巧没有立即遇到适合自己的研究方向。本科时我是参加过国际大学生程序设计竞赛全球总决赛 (ICPC World Finals) 的“职业选手”，也对系统软件颇感兴趣。经过一番考虑（主要是认识到自己数学训练上的不足）之后选择了计算机软件方向。因为之前有一定的算法竞赛背景，所以在导师的帮助下选择了一个和普适计算相关的研究问题。

然而我在第一个研究问题上很久都没有显著的进展。我心中还有一个挥之不去的疑问：我做的这个东西到底有没有用？我想很多博士生都会面临这种困扰，觉得自己在做一些注定要被淘汰的东西，提不起兴趣，仿佛是对自己能力不足的愤慨。颇为流行的调侃是“博士入学的目标是改变世界，但读了几年后，就想赶紧逃离了”。幸运的是我虽然出师不利，但我遇到了好的导师，趁着我“改变世界”的决心还没凉透，在导师吕建教授和张成志教授的帮助下我找到了新的研究方向，也就是在第一部分中提到的并发程序观测相关的问题。

在情绪略有消极的同时，我做了一个（后来看来十分重要的）选择：虽然我的研究问题进展缓慢，但我对计算机科学的憧憬和热情依然没有降低，所以在那个时候我涉猎了很多“无关”的教科书、论文和公开课，也听了很多学术报告。我们系里各个研究方向的老师或许都有印象，听报告次数最多的

那个“外行人”可能就是我了。那个时候我还培养了阅读杂志的习惯，例如阅读每一期 *Communications of the ACM* (*CACM*)，到现在已经有 10 年了，虽然有不少文章并不能完全读懂，但也读到了很多令我惊叹的好文章，也帮助我建立了“其他领域的人在用什么方法解决什么问题”的世界观。我向年轻的博士生们推荐 *CACM* 和《中国计算机学会通讯》(*CCCF*)，其中的文章能带给科研新手很多启发。坚持总是有点困难的，不过之后的回报也的确出人意料。我因为好奇蒂姆·拉夫加登 (Tim Roughgarden) 这样的算法领域的专家如何教入门级的算法课（我是江苏省青少年信息学奥林匹克竞赛委员会的成员，每年有给中学生讲课的任务）而去听了他在 Coursera 上的公开课，因而了解到克罗斯比 (Crosby) 和瓦拉赫 (Wallach) 早在 2003 年发表的算法复杂性攻击的工作 [12]。恰恰是这个机缘巧合，启发我们做了正则表达式复杂性攻击的工作，并且很意外地获得了 ACM SIGSOFT 的杰出论文奖。这段经历让我相信上天总是眷顾那些持续付出的人。

接受挑战

集齐了好的导师和好的研究方向，读个顺风顺水的博士依然是不太可能的。博士研究生面临的典型问题我基本上都遇到过。第一个挑战就是阅读文献。看论文有些类似机器学习中的训练过程：在看到研究问题以后，先试图给出自己的分析和方案，然后再看作者的做法，如果有自己没能想到的奇思妙想，再通过“反向传播”纠正自己思路的盲区。我的导师告诉我，必须阅读 100 篇甚至更多的论文，才能对一个研究领域有一些基础的感觉。我谨记这个教诲，不敢怠慢。令人惊奇的是，后来看论文的速度越来越快了。鉴于选择了一个跨领域的研究问题，坚持不懈使我看到了很多精彩的工作和不同的研究风格——有时候体系结构和程序设计语言研究者看问题的角度很不一样，但在一些细微的地方又不谋而合。

第二个挑战来自做研究的过程。理解了相关的研究工作之后，我好不容易有了一个很棒的想法，后来却发现已经被麦克·邦德 (Mike Bond) 在 OOPSLA'13（ACM 面向对象编程、系统、语言和应用会议）上发表了。一方面感到很受挫，但另一方面自己也挺开心的：能和当前最好的研究工作想到一块去，说明也许还能再坚持一下。于是我想通了：我们诞生的想法，99% 都是别人已经想到或者没有用的（比如实际上实验效果并不好）；但如果能想出 100 个，成功的概率就会很大。所以每当我有新想法时就会记下来，隔段时间再回顾一下，有些可能被否定，有些则可能有新的进展。

想通了这一点，我觉得这才是真正摆正了“读博士”的心态，剩下的就容易多了：只需时间的积累。即便好的想法出现是小概率事件，但只要大量重复，终归还是会有的。所幸我领悟这两点没有花太多时间，加上导师给了我非常宽松的研究氛围，支持我在各种奇思妙想上试错，才顺利完成了博士期间的研究工作。

回归初心

在读博期间我申请了国家留学基金委的访问项目，但因为不知为何的拖延症，联系导师的事项拖到了临近申请截止日期。再一次在导师的大力帮助下，我在很短的时间内联系到了俄亥俄州立大学的秦锋老师，并且成功完成了申请。在联系之初，我对秦锋老师的研究背景了解甚少，只知道和我们的工作有一些交集，对这段访问经历也抱有很大的忐忑。有时候不得不说读博需要一些运气。随后我很惊讶地发现我读过秦锋老师的研究组在 OSDI'14（USENIX 操作系统设计与实现会议，计算机系统领域顶级会议）上发表的一篇有关崩溃一致性的论文，非常喜欢。所以这一方面是运气，另一方面是之前阅读论文的积累派上了用场（这是一篇和我研究方向“完全无关”的论文）。

于是在访问期间我欣然选择了崩溃一致性相关的研究问题。我们做了一个很有趣的小工具：用 ptrace 拦截系统调用，用虚拟设备收集块设备的日志，然后用一些算法上的小技巧来自动检测应用程序中的崩溃一致性缺陷。对于有足够技术储备的我

来说都是轻车熟路，在半年的访问期间就完成了选题、代码、实验，找到了包括 Coreutils、Gzip 等重要软件里的缺陷，论文发表在了 FSE'16（ACM 软件工程基础国际研讨会）上。这些小技巧和我之前的研究关系甚少，也就是在研究起步受阻的时候，做的那些“无用功”派上了用场。最近我收到了 Perl 社区的感谢，在最新的稳定版里修复了我们报告的缺陷。回顾自己求学的经历，技术方面的积累都是互联网和开源软件培养的，自己一直在索取，如今能做出一些微小的回报，欣喜甚至胜过获得博士学位。

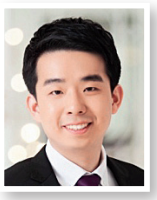
在秦锋老师的指导下，我阅读了很多计算机系统领域的经典论文，有些甚至追溯到 20 年前，若不是他言传身教，自己很难了解研究的历史脉络，其中不乏影响数十年的奠基性论文。秦锋老师还教导我们两件事：“离开你的舒适区 (Get out of your comfort zone)”和“不要轻易放弃 (Don't give up easy)”。这两句话我发自内心地赞同。每次在我的研究似乎山穷水复疑无路时，却又总是能绝处逢生，当然也可能是因为自知天资有限，并敢去尝试那些最难的问题吧。之后我也一直鼓励我的学生，遇到困难的时候给自己一点时间，以自己的经验总能扛过去。因此出去走一走，和不同的人聊一聊，对博士生来说或许是非常重要的。

这一小段经历对我往后的研究方向产生了很大的影响。我不再像刚找到研究方向时只想着“解决点什么问题”，“改变世界”的初心似乎又回来了。在此之后我走上了更实用的研究路线，关注如何从理解需求出发切实地提高软件的质量。我们实现的自动化工具在 Linux 内核和 GCC 等基础性软件上找到了很多前所未知的缺陷。我把现在的研究风格总结成“For fun and profits”，首先为了程序员的那种自我满足，其次是创造一些对社会有用的实际价值。当二者达成一致时，往往能做出不错的研究工作。

感悟

做研究、读博士到底是难还是容易呢？我想二者皆有。计算机科学里很多时候让我们面临多个选择之间的取舍 (trade-off)。挑战困难的问题令人头疼，但即便挑战看似失败，它却依然帮助你成长，给你丰厚的回报——回头看之前觉得困难的问题时，反而觉得容易了。世界就是这么奇妙，我们离理解它还差得远。

走上计算机科学的道路多多少少算是“命中注定”。自幼儿园起就相识的挚友彭泱 (Richard Peng) 和我一起从懵懂开始学习编程，让我总惦记大洋彼岸那个解题神速的身影；大学时代 ICPC 竞赛的队友李昂和李坤帮我树立了对计算机科学的信心和热情；读博期间我的导师吕建、马晓星、许畅教授给了我悉心的指导和无私的支持，才有了我今天的成绩。想要感谢的人难以一一列举，有他们的陪伴才让我始终对计算机科学心存热爱和感激。今天，我站上讲台传授一个软件研究者对计算机系统的理解，也从我的启蒙老师，已离我们远去的江苏省青少年信息学奥林匹克竞赛委员会的李立新教授手中接过接力棒，守护孩子们热爱计算机科学的初心。初心很简单——从来没有一个学科能像计算机科学那样如此快速地让世界变得那么美好，难道不该去珍惜和守护吗？ ■



蒋炎岩

CCF 专业会员、2018CCF 优秀博士学位论文奖获得者。博士毕业于南京大学。南京大学计算机科学与技术系助理研究员。主要研究方向为软件分析、测试与合成。

jyy@nju.edu.cn

参考文献

[1] Gibbons P B, Korach E. Testing Shared Memory[J]. *SIAM Journal on Computing*, 1997, 26(4): 1208-1244.

[2] Xu M, Bodik R, Hill M D. A “Flight Data Recorder” for Enabling Full-System Multiprocessor Deterministic Replay[C]//*Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2003.

[3] Dunlap G W, King S T, Cinar S, et al. ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay[C]//*Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[4] Dunlap G W, Lucchetti D G, Chen P, et al. Execution

Replay for Multiprocessor Virtual Machines[C]// Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE), 2008.

[5] Bond M D, Kulkarni M, Cao M, et al. Octet: Capturing and Controlling Cross-Thread Dependences Efficiently[C]//Proceedings of the ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA), 2013.

[6] Huang J, Zhang C, Dolby J. CLAP: Recording Local Executions to Reproduce Concurrency Failures[C]// Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2013.

[7] Jiang Y, Xu C, Li D, et al. Online Shared Memory Dependence Reduction via Bisectional Coordination[C]// Proceedings of the International Symposium on the Foundations of Software Engineering (FSE), 2016.

[8] Chen D, Jiang Y, Xu C, et al. Testing Multithreaded Programs via Thread Speed Control[C]. //Proceedings of the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2018.

[9] 蒋炎岩 , 许畅 , 马晓星 , 吕建 . 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 , 28(4):747-763, 2017.

[10]Attiya H, Guerraoui R, Hendler D, et al. Laws of Order: Expensive Synchronization in Concurrent Algorithms Cannot be Eliminated[C]//Proceedings of the ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages (POPL), 2011.

[11]Spielman D A, Teng S H. Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time[J]. Journal of the ACM, 51(3), 2004: 385-463.

[12]Crosby S A, Wallach D S. Denial of Service via Algorithmic Complexity Attacks[C]// Proceedings of the USENIX Security Symposium (USENIX Security), 2003.