★CACM 中国版★

计算机协会通讯

CACM.ACM.ORG

2014 年 3 月第 57 卷第 03 期

隐写术

在数据中隐藏数据

APPLY HEAT

失败的研究中心是如何建成的

API的性能约定

使并行程序变得可靠

TaintDroid

Association for Computing Machinery

ACM 通讯

精选文章中文译文

ACM通讯中国版编辑

主席

陈文光
清华大学
cwg@tsinghua.edu.cn
并行计算和编程语言
陈文光教授现任清华大学计算机科学与技术系教授、副主任。

委员

陈海波
上海交通大学
haibochen@sjtu.edu.cn
操作系统和计算机体系结构
陈海波教授就职于上海交通大学软件学院。

崔斌
北京大学
bin.cui@pku.edu.cn
数据库
崔斌教授就职于北京大学息科学技术学院，并担任网络与信息系统研究副所长。

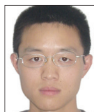
陈贵海
上海交通大学
gchen@cs.sjtu.edu.cn
上海交通大学计算机科学与工程系教授；中国计算机学会开放系统专委会主任；在并行与分布式计算领域有广泛的兴趣，特别是各种网络系统，例如无线传感器网络，对等覆盖网络，数据中心网络，社交网络等。

李向阳
伊利诺理工学院
xli@cs.iit.edu
李向阳教授就职于伊利诺理工学院。他是中国国家科学基金会海外杰出青年学者奖的获得者。

刘云浩
清华大学
移动和普适计算，RFID和传感器网络系统
yunhao@greenorbs.com
刘云浩教授现任清华大学长江特聘教授。他还担任ACM中国理事会主席。

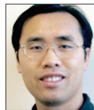
山世光
计算技术研究所
sgshan@ict.ac.cn
计算机视觉和图案识别
山世光教授就职于中国科学院计算技术研究所（ICT）。

孙晓明
计算技术研究所
sunxiaoming@ict.ac.cn
理论
孙晓明教授就职于中国科学院计算技术研究所。

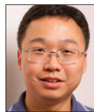


唐杰
清华大学
jietang@tsinghua.edu.cn
数据挖掘
唐杰副教授就职于清华大学计算机科学与技术系。

田丰
*中国科学院*软件研究所
tianfeng@iscas.ac.cn
人机交互
田丰教授就职于中国科学院软件研究所，他还担任计算机协会中国人机交互学会主席。



谢涛
伊利诺伊大学厄巴纳–香槟分校
taoxie@illinois.edu
软件工程
谢涛副教授就职于美国伊利诺伊大学厄巴纳－香槟分校计算机科学系。



周昆
浙江大学
kunzhou@acm.org
计算机图形和虚拟现实
周昆教授是长江特聘教授，浙江大学CAD&CG国家重点实验室主任。

诸葛建伟
清华大学
zhugejw@cernet.edu.cn
计算机安全
诸葛建伟副教授就职于清华大学网络科学与网络空间研究院。

计算机协会中国理事会
孙家广，名誉主席
刘云浩，主席
沈运申，副主席，分会
陈文光，副主席，出版物
王新兵，副主席，会议
万猛，副主席，宣传与公共关系
张铭，常务理事
肖人毅，常务理事
吕自成，常务理事
秦志光，常务理事
罗军舟，常务理事
胡传平，常务理事
胡斌，常务理事
赵峰，常务理事

计算机协会中国顾问委员会
孙家广，联席主席
李志民，联席主席
姚期智
廖湘科
王珊
怀进鹏
梅宏
吕健
郑南宁
张尧学
林惠民

分会主席
上海分会 胡传平
南京分会 罗军舟
成都分会 秦志光
兰州分会 胡　斌
重庆分会 廖晓峰
长沙分会 卢　凯
广州分会 张　军
济南分会 杨　波

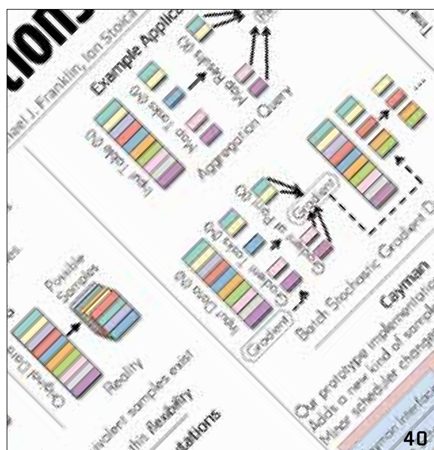
ACM中国理事会
中国北京清华大学
东主楼 11-236 室
邮编：100084
电话：+86-10-62785025
电子邮件：acmchina@acm.org
联系人：辛爽

ACM通讯
(ISSN 0001-0782) 由计算机协会（2 Penn Plaza, Suite 701, New York, NY 10121-0701）按月发行。

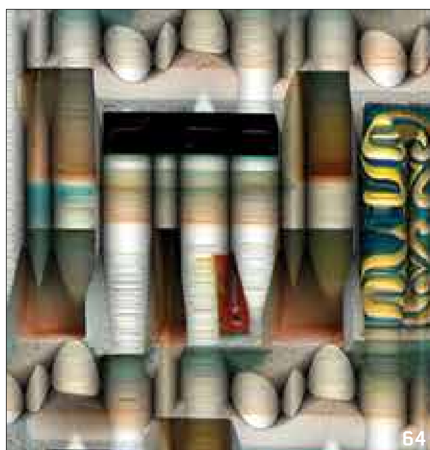
Association for Computing Machinery

3/2014
第 57 卷 第 3 期

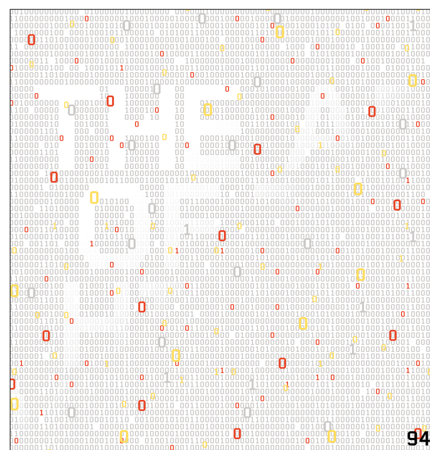
观点



投稿文章



评论文章



33 **观点**
失败的研究中心是如何建成的
分享从创建成功的跨学科研究中心的经历中学到的教训。
作者：David Patterson

58 **利用稳定多线程使并行程序变得可靠**
稳定多线程大大简化并行程序的交叠行为，有望使并行编程更加容易。
作者：Junfeng Yang、Heming Cui、Jingyue Wu、Yang Tang 与 Gang Hu

86 **隐写术的趋势**
与古代相比，如今秘密数据的嵌入方法更加复杂，不过基本原理依然没变。
作者：Elzbieta ZieliNska、Wojciech Mazurczyk 与 Krzysztof Szczypiorski

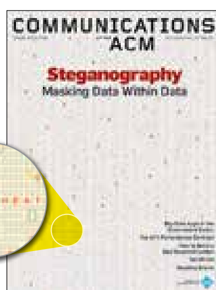
实践

45 **API的性能约定**
如何确保调用者与实现之间的期望交互？
作者：Robert F. Sproull与Jim Waldo

研究亮点

98 **技术视角**
"污点跟踪"的前世今生
作者：Dan Wallach

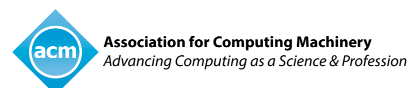
99 **TaintDroid：用于在智能手机上实时监控隐私数据的信息流跟踪系统**
作者：William Enck、Peter Gilbert、Byung-Gon Chun、Landon P. Cox、Jaeyeon Jung、Patrick McDaniel 与 Anmol N. Sheth



关于封面：

本月的封面故事探讨了人类历史上的信息隐藏技术。隐写术可追溯至古希腊，在那里秘密信息的嵌入方法得以成型。如今，数据可 隐藏在数字媒体、以及诸如 热显影墨水 等载体中。

交互式封面设计：Andrij Borys Associates



图片由AMPLAB/UC BERKELEY惠赠；插图由JOEL ORMSBY提供；图片由ANDRIJ BORYS ASSOCIATES提供

DOI:10.1145/2566969

David Patterson

视角
失败的研究中心是如何建成的

分享从创建成功的跨学科研究中心的经历中学到的教训

一个大型研究中心的建设工作刚刚结束[10]，我终于有时间来总结一下关于研究中心建设的想法。我曾是十几个计算机研究中心的一员，并且很多时候是那里的主任（详见附表）。这里所说的研究中心，是指拥有至少三位科研人员，十几个学生，以及一个共同目标的研究项目。本文的观点是从计算机系统的科研人员角度出发的，但我希望它能有更广泛的应用，甚至扩展到学术以外。我并不主张所有的研究都要设立研究中心；对很多课题来说，只要一个研究员就非常好。为什么要重视伯克利的经验呢？唉，整个说服过程很像是在吹牛，因此我提前表示歉意，稍后我将会再次致歉。*美国自2002年以来《美国新闻与世界报道》（News and World Report）*已经先后四次对相关大学就计算机领域进行排名。每一次，我们的同行们都将加州大学伯克利分校排在第一。此外，美国国家研究委员会（National Research Council）针对催生了诸多数十亿美元产业的信息技术研究，公布了一份报告。[6]加州大学伯克利分校与其中的7个产业有关，数量位于所有大学之首，主要就是因为它的科研项目。

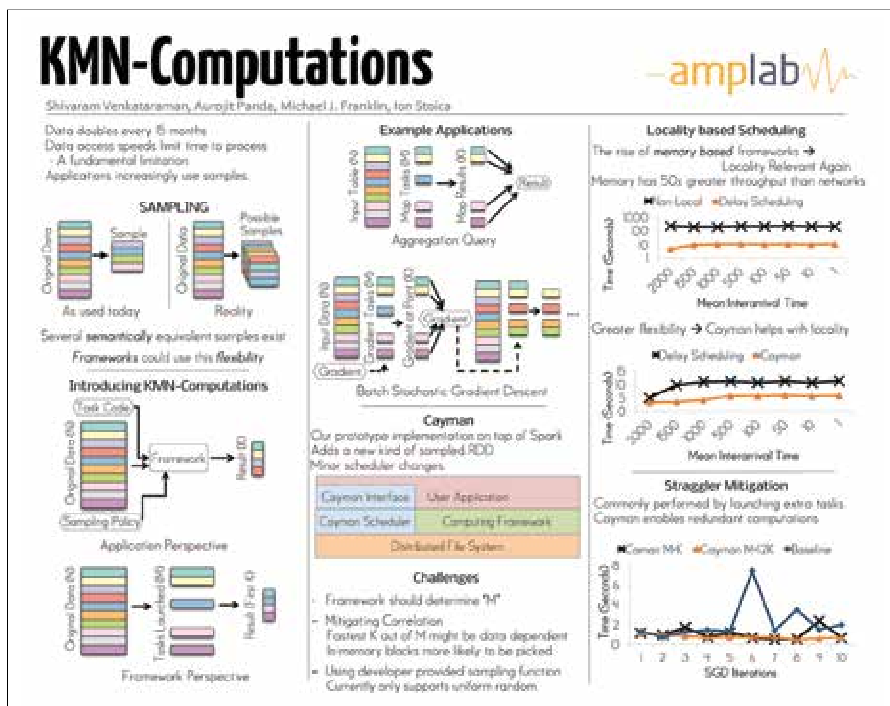
成为一个失败的研究中心的八大铁律

沿袭我早期文章的路子[a, b]我总结了“失败的研究中心是怎样建成的”之八大铁律。稍后我会介绍如何避免成为失败的研究中心。

失败的铁律之一您不该在一个中心里混杂不同的学科不同学科背景的人之间的相互交流是很困难的，因为他们之间并没有共同的文化和语言。因此，多学科混杂不仅浪费时间，还会挥霍宝贵的科研经费。相反地，应该保持学科单一。

失败的铁律之二您要扩展科研中心。这种扩展应从地理位置，而非人才角度加以考量。比如在美国的话，理想的做法是从全美五十个州的五十个科研机构招募研究者，这样可以让资助机构在美国参议院面前有光。

失败的铁律之三您不该限制科研中心的周期。为了表明你对科研中心使命的信心，你应当愿意投身这个项目数十年。（或者至少会持续到资助资金告罄。）



在加州大学伯克利分校AMPLab open house召开的2013伯克利EECS年度研究研讨会上的海报样例。

图片由 AMPLAB/UC BERKELEY惠赠

a D. Patterson, http://www.cs.berkeley.edu/~pattrsn/talks/BadTalk.pdf, 1983.

b D. Patterson, http://www.cs.berkeley.edu/~pattrsn/talks/BadCareer.pdf, 1997.

失败的铁律之四您不该建立一个精雕细琢的原型。 将各种结果整合到一个科研中心级别的原型中，势必会消耗科研人员本人的时间，更重要地是，进而影响科研人员的个人研究。

失败的铁律之五您不该打扰同事。 好的隔离设施造就好的研究者；隔离减少了工作分心的情况。

失败的铁律之六您不该与科研中心以外的人交流。 不要浪费时间去召开会议去把研究展示给科研中心以外的人；根据第8条诫言，各类论文受到的评论就能为你提供足够的反馈。

失败的铁律之七您要通过与同仁达成共识而作出决策。 美国国会就是一个通过共识而取得进展的典型示例。

失败的铁律之八您要尊重论文出版机构。 科研人员的工作成效按其所著论文数量与论文引用次数加以考量。因此，为确保研究中心取得成功，你必须不停地撰写论文，论文不停地获引用。如果会议的论文录取率是1/X, 那么显然，你应该至少提交X份论文，否则有可能你的科研中心在有些会议上没有发表论文，这将是一场大灾难。

失败的研究中心重获新生

失败研究中心要想重获新生需要将上述“八大铁律”统统破除。就拿我在加州大学伯克利分校的亲身经历来讲，我向附表中的同事们做过调查，结果发现卡内基梅隆大学、谷歌、哈佛大学、威斯康星大学、加州大学圣地亚哥分校的研究项目也正在打破这些铁律。

成功的建言之一您要融合多个学科。

成功的建言之二 您要限制中心的自身规模。 底层技术日新月异，为我们的研究领域带来了诸多新的契机。虽然企业比我们拥有更多的资源，但他们在传统层面进行创新却是困难重重。他人给予的心理支持也能够鼓舞团体的士气。因此，相对于个人和企业，如果愿意持续吸引非计算机科学的学科加入，跨学科的团队将会拥有更好的机会。我相信在这些快速发展的领域，学科间相互碰撞、相互作用的机会要大于学科内部。

申请人为增加获得经费的机会往往不惜作出任何承诺，却忽视在获得经费后对中心的运作。例如，很多人认为研究中心拥有的科研人员和研究机构数量越多越好，这样可以提升申请经费的成功率。很简单的一个理由就是评估科研的难度；由于评价研究中心的影响力需要不少时间，因此庞大的研究中心不会对项目申请产生不利影响。下面的研究结果提示我们应该遵守以上两条对我们有益的建言。研究人员在对由美国国家科学基金会（NSF）资助的62所计算机科学研究中心进行考察后发现，多学科能增加研究项目的成功率，而研究项目由多个研究机构（特别是研究机构数目非常大）共同完成的话，则会降低成功率：“平均来说，由单一大学开展的项目要比我们研究过的多所大学共同完成的项目更加成功......涉及跨学科的研究项目在同一所大学内进行时成效更佳。”[2]

跨学科研究的一个不利因素就是需要花时间去理解文化和词汇上的差异。但是，如果你相信跨学科研究中心是提高影响力的最佳途径，那么收益就会大于支出。

成功的建言之三您应该限制中心的周期。 如附表所示，我的事业就基于这些5年周期的研究中心。基于以下三种观点制订了这条五年的届满条款：

- *要多打全垒打我们就要多上场击球*。幸运的是，人们记住的是研究的“全垒打”而不是几近成功的情况。我的经验就是与其说研究的最终目标的实现是靠多年的研究付出，不如说是靠多个研究项目的共同产出，因此，短期项目更容易做出贡献。
- *很难预计5年以后信息技术的发展趋势*。基于对“未来7-10年内会有哪些机遇” —— 一个5年研究中心的正确目标——的最佳评估，我们启动了这个中心。15-20年内会有哪些机遇就极端的难以预测了，这需要更长周期的项目研究。
- *在美国，研究生的修学年限一般为5年*。 如果没有人员流动的话，运作一个中心会容易得多。由于我们处于学术环境中，每个新项目启动时我们都将招收一批新学生，然后要过五年他们才能毕业。

在一个中心结束后，我们需要约十年才能判断其是否算大获成功。附表中列出的12个中心里面仅有8个历时久远，但其中只有三

David Patterson任职过的研究中心（研究中心主任在第三列列出）。项目规模能否随时间扩大，既取决于能否持续成功地筹集资金，也取决于是否尝试解决越来越大的研究问题。

年份	项目名称	教授：主任，课题负责人	学生
1977–1981	X-Tree:Tree Multiprocessor	Despain, Patterson, Sequin	12
1980–1984	RISC:Reduced Instructions	Patterson, Ousterhout, Sequin	17
1983–1986	SOAR:Smalltalk On A RISC	Patterson, Ousterhout	22
1985–1989	SPUR:Symbolic Processing Using RISCs	Patterson, Fateman, Hilfinger, Hodges, Katz, Ousterhout	21
1988–1992	RAID:Redundant Array of Inexpensive Disks	Katz, Ousterhout, Patterson, Stonebraker	16
1993–1998	NOW:Network of Workstations	Culler, Anderson, Brewer, Patterson	25
1997–2002	IRAM:Intelligent RAM	Patterson, Kubiatowicz, Wawrzynek, Yelick	12
2001–2005	ROC:Recovery Oriented Computing Systems	Patterson, Fox	11
2005–2011	RAD Lab:Reliable Adaptive Distributed Computing Lab	Patterson, Fox, Jordan, Joseph, Katz, Shenker, Stoica	30
2007–2013	Par Lab:Parallel Computing Lab	Patterson, Asanovic, Demmel, Fox, Keutzer, Kubiatowicz, Sen, Yelick	40
2011–2017	AMP Lab:Algorithms, Machines, and People	Franklin, Jordan, Joseph, Katz, Patterson, Recht, Shenker, Stoica	40
2013–2018	ASPIRE Lab	Asanovic, Alon, Bachrach, Demmel, Fox, Keutzer, Nikolic, Patterson, Sen, Wawrzynek	40

个——RISC、RAID和NOW[c]研究中心——可以被认为是“大获成功”的。如果37.5%的成功率就算好的话，那么我宁愿有很多五年的研究中心，而不是少数几个周期较长的中心。

届满条款的一个不利之处在于，随着中心声誉日隆，招收学生和延揽资金会比新中心更容易。然而，如果目标是尽可能增加“大获成功”的研究中心数，而非招人或找钱，那么在我们飞速变化的领域内，最好五年后就宣布胜利，并且重新寻找新机遇，然后招收合适的团队去实现。

成功的建言之四您 要 建立一个中心级的原型。附表中的各个研究中心都有一个共性，那就是它们都建立了一个展示中心研究目标的原型，这个目标可以确保中心各个机构共生共存。尽管计算机系统内的学生喜欢为中心的研究目标有所建树，但他们不一定会契合各个方面去跟其他人合作。然而，不同背景的学生在一起工作，就使得跨学科研究中心有了教育能力，科研实践会促使他们提高对研究的理解和研究的品位。需要学生去做真正的原型系统而非小型实验展示，这样的过程同样会提高学生的系统构建能力。原型系统研究甚至可以引领开源项目，这也有助技术转让，和将创建原型的工作扩展开来。实际上，成功的开源意味着更多的外部开发者而非内部开发者。

这样的研究中心的一个不利之处在于老师们必须使学生明白花时间构建共用原型而不是满足个人研究兴趣的好处所在。然而，一旦他们开始发现别人的研究对他们自己的研究有帮助，他们就再无需鼓励。

成功的建言之五您要打扰您的同事。研究发现，所有人在一个50米范围内的开放空间内工作会提高创新能力，因为这样会促进自发的跨学科交流。[1]开放空间的目的就在于既可以支持集中研究又可以增强相互交流。例如，附表中的后4所研究中心，科研人员放弃私人办公室，选择和学生、博士后一块在一个开放空间工作。在这个开放空间中，只有会议室才有墙。[8]能够接触科研人员的机会将学生从家里吸引到实验室来，增加了交流机会。

我相信在这些快速发展的领域，相比较单一学科研究，夸学科的研究会更加有机会做出有影响力的东西。

共享研究空间的不利之处在于创建一个有吸引力的开放空间的花费。 这是一个一次性的投资，因为接下来的项目可以继续使用这个场地。但即使这样，这个费用也不过等于研究中心运转期间在两个学生身上的开销。共享的研究空间对中心的好处明显大于几个额外的学生带来的好处。

成功的建言之六您要 和不熟悉的人去交谈。研究中心成功的关键之一在于外部反馈。每年我们举办两次所有中心成员和其他科研机构数十人参加的务虚会，每次会议持续三天时间。这好比是设置一个“程序委员会”，该委员会的职责就是五年中每年有六天时间与中心每个人会晤，或者每个中心每五年有一个月的反馈期。务虚会的价值已经100%的为我的同僚们所验证。

与外机构人员长时间的讨论经常会重塑学生和科研人员的视角，因为这些客人带来了全新的观点。更为重要的是，每次务虚会后我们从贵宾学者那里都会获得开诚布公的，有信息量的，细致的反馈意见。独立研究者需要深入的有建设性的批评意见，但他们很难听到这些建议。在务虚会上我们不允许当场反驳反馈意见；相反，我们在会后会仔细地考虑这些意见。实际上，我们说的是我们会严肃地听取来宾的意见，所以别惊讶你收到的各种建议。

务虚会还有其他重要的功能：

- 这可以提供真正的里程碑，这在学术界是很罕见的。可能答应你的导师在一月份将会做完一些什么事情比较简单，但是当你计划和别的到访者来谈谈你们的研究时，这就变成了另外一件比较大的事了。
- 他们会组建一个强有力的团队，中心的每一个人都会在一起呆上3天。工作之余学生们会在一起娱乐，所以他们后来都成了终身的朋友。实际上，通过调查我非常高兴的知道，由于他们共同的经历，不同时期的中心的校友们还是保持紧密的联系。
- 在5年的研究中心生涯中，所有的学生都有10次海报展示或报告的机会。这给了他们与渴望的一样多的自我展示机会，还提高了他们的交流技巧。

务虚会的不足之处同样是花费较高，费用大概是培养一到两个学生所花的费用。 如上所述不组织务虚会而多培养几个学生是得不偿失的。邀请外面的同行参与务虚会时要确保他们能够全身心投入并提出有用的建设性意见。

成功的建言之七您要找到一个好领导。为推动项目进展，您需要找到这样一个人：愿意花时间将大家聚到一起、树立一个共同的愿景、打造团队精神，并且大家都相信他做决策时能以研究中心利益为先。 作为中心主任，我通常为团队的成功进行博弈而不仅仅是位著名的教练。因此，我招募的科研人员都具有团队协作精神，自命不凡的家伙则留给别人。我用以身作则的方式来领导团队，为了中心的成功努力工作， 希望藉此能够影响团队成员。我的信条是“成功的团队没有失败者，失败的团队没有成功者”。

成功的建言之八您要尊重*影响*力。尽管论文发表确实是反映高影响力的研究，但它们并非考量研究成功与否的直接标准。高影响力的研究成果甚至很难公开发表。2012年ACM软件系统大奖的获得者、关于LLVM编译器的第一篇论文，最

c NOW项目展示了集群工作站能够为从加密到索引提供全部服务。NOW加载了Inktomi搜索引擎。刚刚起步的Inktomi公司接下来证明了许多廉价计算机组建的集群的价值要高于少量高端服务器，谷歌和其它互联网企业后来跟进。

初被主要的编译器研讨大会PLDI拒绝，而蒂姆·伯纳斯·李（Tim Berners-Lee）关于万维网的第一篇论文也被Hypertext'91拒绝。

最佳论文奖也并非高影响力的指标。有一项研究统计了最佳论文的被引用次数，其中大多数文章只有较低影响力。。[d]获得2012 ACM InfoSys大奖的、第一篇关于MapReduce的论文，并没有获得最佳论文奖。当时的程序委员会将最佳论文奖颁发给了另外两篇论文！会议结束10多年以后才颁发的“时间考验奖”，是一个评判研究成功与否的更好的指标，但这样的指标却很滞后。

在我的成功的项目中，早期指标并非论文或奖项，而是普通用户想用我们的想法和技术。 对RISC和NOW项目，最初的使用者是新兴公司或者至少是初创不久的公司。对RAID项目而言，其用户则是对占有市场份额如饥似渴的众多公司。市场的主流是对现状最好的诠释；只有技术取得商业成功 *之后*市场才会开发其相关产品。

研究中心的教育影响力

研究中心能够提高大学本科教育质量，这原本是研究型大学的宗旨之一，但有些人对此颇为诧异。在RISC项目中看到微处理器流水作业之后，不到10年内，大学普遍教授本科学生自行设计微处理器。是否应该为计算机系学生开设软件工程课程一直众议纷纷，RAD实验室则克隆了我们的软件工程课程。[3]伯克利分校选课学生人数从35人飙升至240人，2012年通过大量在线公开课程获得这门课程证书者有一万多人。 这些课堂改革促生了三本教材，进一步扩大了教育影响力。[4,5,9]

良好的研究中心是诸多子项目的联盟，这些子项目共享一个相同的研究目标，每个子项目都有一到两位教授和若干学生。每位教授都是不同领域的专家，有力地促进着跨学科研究。类似独立研究的项目，这些子项目让学生获得相同的教授关注机会，此外学生还可以：

- 通过和数十位聪明、勤奋、固执的合作者（有些人甚至是出了名的执拗）一起工作和商讨来学习新知。
- 对一个共同的原型进行研究而获得美好的科研体验，并产出让人耳目一新的学位论文。
- 获得新工具和新方法的用户群和反馈。
- 在一个开放空间有学长学姐作为榜样，可以获得明确的指导。
- 来自于开放空间和务虚会的团队精神会使博士阶段那种普遍的孤独感荡然无存。
- 务虚会专家的赞扬将使你重新斗志昂扬。

在这种成功的模式面前我一点也不谦虚，就如我之前所说，为此我再次事先致歉。在过去十年DARPA削减大学科研经费的时候，[7]我们给公司提出的议案是我们需要跨学科研究中心来继续系统化培养优秀的计算机学科的学生。很高兴，我们的主张被证明是正确的。比如，在RADs实验室公共场所工作的第一批学生中就有一个学生，他在偶尔听到实验室里使用机器学习的学生在抱怨MapReduce时产生了Spark集群计算框架的想法。[11]为此他不仅收到了他所访问过的公司就职邀请函，也收到了所有顶尖大学的聘用书。

你稍加留意的话就会发现，这些项目不止产生一颗巨星。比如，Par实验室平均每篇论文有三位学生，每个学生平均有三篇高质量论文。更重要的是，伯顿·史密斯（Burton Smith）评价说：“他们是我见过的最好的研究生。”这可是来自微软研究员和美国国家工程院院士的高度评价。史密斯先生在2013年的评价固然让人倍受鼓舞，但我却被普适计算的先驱马克·维瑟（Mark Weiser）在1989年的同样评价SPUR学生的话语所震惊。

跨学科研究中心的五年思考

如果这样的研究中心对科研人员和学生都有利，那我们就可以告诉大家。一个衡量标准就是美国国家工程院和美国国家科学院的选举：每年只有5-10名计算机科学家入选工程院院士（一半来自工业界），仅有2-3名入选科学院院士。而迄今为止，供职于附表所列研究中心的24位伯克利计算机科学研究人员中有9位是工程院院士（比例约为40%），其中3位同时还是科学院院士。实际上，大部分伯克利分校计算机教授在美国国家工程院和科学院的院士从事上述项目的科研工作。 实际上，伯克利分校大部分被选为美国国家工程院和科学院院士的计算机教授都在从事上述项目的科研工作。更让你惊讶的是，他们都是在加入这些研究中心之后当选院士的。

尽管早期的计算问题可能是由单一学科的单一科研人员解决的，但我相信需要跨学科团队的计算问题比重将会增加。如果是这样，比之过去的40年，在接下来的40年内学习如何组建运作良好的研究中心将会更加重要。 C

d 参见Best Papers vs.Top Cited Papers in Computer Science; http://arnetminer.org/conferencebestpapers.

参考资料

1. Allen, T. and Henn, G. *The Organization and Architecture of Innovation:Managing the Flow of Technology*.Butterworth-Heinemann, 2006.
2. Cummings, J. and Kiesler, S. Collaborative research across disciplinary and organizational boundaries. *Social Studies of Science 35*, 5 (2005), 703–722.
3. Fox, A. and Patterson, D. Crossing the software education chasm.*Commun.ACM 55*, 5 (May 2012), 44–49.
4. Fox, A. and Patterson, D. *Engineering Software as a Service:An Agile Approach Using Cloud Computing*, First Edition.Strawberry Canyon, 2014.
5. Hennessy, J. and Patterson, D. *Computer Architecture:A Quantitative Approach*.Fifth Edition. Morgan Kaufmann, 2011.
6. *Innovation in Information Technology*.National Research Council Press, 2003.
7. Lazowska, E. and Patterson, D. An endless frontier postponed.*Science 308*, 5723 (2005), 757.
8. Patterson, D. Your students are your legacy.*Commun. ACM 52*, 3 (Mar. 2009), 30–33.
9. Patterson, D. and Hennessy, J. *Computer Organization and Design:The Hardware/Software Interface*.Fifth Edition.Morgan Kaufmann, 2013.
10. *The Berkeley Par Lab:Progress in the Parallel Computing Landscape*. D. Patterson, D. Gannon, and M. Wrinn, Eds., 2013.
11. Zaharia, M. et al.Spark:Cluster computing with working sets.In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing* (2010).

David Patterson (pattrsn@cs.berkeley.edu)是美国加州大学伯克利分校的E.H. 和M.E. Pardee Chair，他曾任ACM主席。

译文责任编委：李向阳

版权归属于作者 / 所有者

David Patterson先生访谈见112页.

DOI:10.1145/2576792

文章编写主导者 acmqueue
queue.acm.org

如何确保调用者与实现之间的期望交互？

ROBERT F. SPROULL 和 JIM WALDO

API的性能约定

调用API函数时，你期望它们执行正确；在有些场合中，这种期望被称为调用者与实现之间的约定。调用者对函数也有性能方面的期望。通常而言，软件系统能否成功同API满足这些期望的能力息息相关。

所以，除了***正确性约定***外，还有一种***性能 约定*** 。性能约定一般都不明确，通常比较含糊，（调用者或实现者）有时候还会违反约定。那么， 我们怎样才能在API的设计和文档中改进这一方面？

现在，所有重要的软件系统均依赖于其他系统的工作：你当然需要写点代码，但是你可以通过API调用操作系统的函数或各种软件包的函数来减少你自己代码编写的数量。在有些情况下，你甚至要通过不稳定的网络把工作外包到远程的服务器上去。你依赖这些函数和服务实现正确的操作，而且你也依赖它们的性能。它们的性能要够好，这样整个系统才能运行良好。在复杂的系统中，因为涉及分页、网络延迟、资源（如磁盘）共享等因素，必定会有一些性能的变化。即使在简单的设置中，比如把所有程序和数据载入内存的单机中，你也会惊奇发现API或操作系统有时没有达到期望性能。

当API函数被调用时，人们习惯于用API的应用和实现之间的约定来描述调用的正确行为。调用者必须满足特定的初始条件，而函数则按照规定执行。（这种双重责任有点像霍尔逻辑中出现的前置条件和后置条件。霍尔逻辑用于证明程序的正确性）。在现在的API规范中，虽然正确性的判断标准描述得不够清晰，不足以让人据此证明正确性，但在API函数的类型声明和文本文档中，人们还是尽量把函数的逻辑行为描述得清晰无误。如果人们描述某个函数为“在链表末端

*l*添加元素*e*”，并在其调用序列中描述了*e*和*l*，那么调用者便能了解调用会引发的行为。

不过，对于API函数来说，仅仅正确还不够。它会消耗哪些资源？它又有多快呢？人们通常会自己判断应该如何实现被执行的函数，然后根据自己的判断做一些假设。在链表后面添加一个元素的代价应该比较“低廉”。逆置链表的代价可能为“链表长度的线性函数”。对于众多简单的函数来说，这些直觉足以避免麻烦 - 虽然有时会不准，如同本文随后描述的那样。然而，对于复杂度迥异的函数而言，比如“在窗口*w*的（*x*,*y*）处，*f* 前绘出字符串*s*”，或者“找出保存在远程文件中数值的平均值”，性能有可能让人惊喜，或让人失望。不仅如此，API的文档中丝毫不会提及哪些函数低廉，哪些昂贵。

更复杂的情况是，依据API的性能特点调优应用后，API的实现出了新版本，或者使用了新的远程服务器保存数据文件，结果总体性能不但没有提升（这是对新事物的永恒期望），反而更糟。总之，在系统中，与软件构件的组成相关的性能约定值得大家更多的关注。

某些库为函数提供了不止一种执行方法，通常是因为各种执行方法的性能迥异。

性能的分类法

在早期，程序员开始构建基于直觉的API性能模型（见侧边栏1）模型需要有用，但不需要很精确。下文说明了一个简单的分类法：

一直低廉（例如：toupper，isdigit，java.util.HashMap.get）。前两个函数的开销一直低廉，通常用的是内联表查询。在大小适当的哈希表中进行查询应当快，但哈希冲突可能会使某些访问变慢。

通常低廉（例如：fgetc，java.util.HashMap.put）。在很多函数的设计中，大多数时候快，但也偶尔会调用更复杂的代码；fgetc有时候必须读取新的字符缓冲区。在哈希表中存入新元素可能会让哈希表变满，实现者会扩大哈希表，然后重新处理所有的元素。

java.util.HashMap 的文档中说明了性能约定，开了个好头：“此实现假定哈希函数将元素适当地分布在各桶之间，可为基本操作（get 和put）提供时间恒定的性能。迭代 集合（collection ）视图所需的时间与 HashMap 的“容量”成正比……”[3]

fgetc的性能则由基础流的属性决定。如果其为磁盘文件，则函数通常会读用户内存的缓冲区，而不需要调用操作系统，但是有时候它必须调用操作系统来读入新的缓冲区。如果它读入的是键盘输入，则实现可能需要调用操作系统，读取每一个输入的字符。

可预测（例如：qsort，regexec）这些函数的性能因其参数的属性不同而异（例如，要排序的数组的大小或者要查找的字符串的长度）这些函数通常属于数据结构，或者通用的算法工具，它们使用了众所周知的算法，不需要进行系统调用。通常情况下，你可以根据对底层算法的期望判断性能（比如，那种排序花费的时间为$n \log n$）。但是，如果使用了复杂的数据结构（比如B-树）或泛型集合（此时可能很难确定底层的具体实现），估算性能可能更难。可预测性只不过是一种概略的判断，理解这点相当重要；比如，通常可以根据输入预测regexec的性能，但是一些病态表达式却会造成时间呈指数级暴增。

无法预测（例如：fopen，fseek，pthread_create，很多“初始化”函数，以及遍历网络的调用）。这些函数不低廉，性能往往也变化很大。它们从资源池（线程、内存、磁盘、操作系统对象）中分配资源，一般需要独占访问共享的操作系统或I/O（输入输出）资源。一般情况下，初始化需要的开销都较大。如果调用需要通过网

络，那就无法避免巨大的开销（相对与本地访问而言），而且开销的变化程度可能更为严重，这使得构建合理的性能模型变得难上加难。

线程库是性能问题的一个直观例子。经过多年的努力，POSIX标准总算尘埃落定，可是实现仍然受到诸多问题的困扰。[6]多线程应用的可移植性仍然充满风险，无法预测。导致多线程出现困难的一些原因如下：需要与操作系统紧密集成，但几乎所有的操作系统（包括Unix和Linux）在设计之初均未考虑线程问题；与其他库进行互动，特别是要让函数变成线程安全的，并处理由此引发的性能问题;以及多线程实现使用了多种不同的设计要点，其大致可分为轻量级和重量级。

按性能将API归类

某些库为函数提供了不止一种执行方法，通常是因为各种执行方法的性能迥异。

让我们回到侧边栏1，我们注意到，大多数程序员都被告知，如果需要获取每一个字符，使用库函数并非最快的方式（即使把函数的代码内联，避免了函数调用的开销后，情况也是如此）。更注重性能的方法则是，读取超大的一组字符，然后利用编程语言中的数组或指针操作抽取每个字符。在极端的场景中，应用可以把文件的分页映射到内存的分页中，从而避免把数据复制到数组的操作。作为性能的补偿，这些函数提高了对调用者的要求（比如，为了得到算法正确的缓冲区，并且让实现与库的其他调用（比如fseek的调用）保持一致，需要调整缓冲区的指针，甚至内容）。

人们一直忠告程序员，不要过早优化程序，程序员因此推迟了极端的修改，直至事实证明较简单的办法不够用。确定性能的唯一方法是对其进行度量。然而，通常在写完了整个程序之后，程序员才发现性能的期望（或估算）与实现者提供的现实之间存在差距。

侧边栏 1.

构建自己的性能模型

通过学习如何编程，你可以非常早地获取性能方面的经验法则。下面的伪代码说明了一种模式，可用于合理有效地处理中等大小的字符文件：

```
fs = fopen("~dan/weather-data.txt", "r");  //(1)
for ( i=0; i<10000; i++) {
   ch = fgetc(fs);                        //(2)
   // 处理字符ch
}
. . .
```

在人们的期望中，函数调用（1）需要一点时间；但是读取字符的调用（2）应该低廉。从直觉上来说，这有点道理：处理文件时，只要打开一次流，但是却需要经常调用“读取下一个字符”的函数，也许是成千上万次，或是上百万次。

有一个库实现了这两个流函数。库[2]的文档中清楚地描述了这些函数的作用 - 用非正式的方式说明了实现与应用之间的正确性约定。其中未提及性能，也没有给程序员提供任何暗示，指出两种函数的性能差异巨大。基于这种情况，程序员根据经验，而不是规范来构建模型。[7]

并非所有的函数都会引发明显的性能属性。例如：

```
fseek(fs, ptr, SEEK _ SET); //(3)
```

倘若目标文件的数据已经放入缓冲区，此函数可能低廉。但在一般情况下，此函数需要调用操作系统，有时候还涉及I/O。在极端的情况下，它可能需要卷出几千英尺的磁带。还有一种可能，就是即使在简单的场景中，库函数也不低廉：实现者可能只是简单的保存下指针，然后放上一个标志，把困难的工作留给读取或写入数据的下一次流调用完成。这样便把性能的不确定性推给了另一类低廉的函数。

侧边栏 2.

函数蔓延

```
SetFontSize(f, 10);
SetDrawPosition(w, 200, 20);
DrawText(w, f, "This is a passage.");
```

此例源于一个虚构的窗口系统：设置字体大小以及绘制的位置，然后再绘制一些文字。你可能期望所有这些函数都相当低廉，因为在屏幕上渲染文字窗口相当快。实际上，在早期的视窗系统，比如Macintosh上的QuickDraw中，你的期望可能是对的。

公正的来说，现在的视窗系统的功能都悄悄地向上层发展了。字符的光栅- 即使是那些在显示器上渲染的字符光栅- 也是通过几何轮廓计算得来，因此准备一种特定大小的字体可能需要一点时间。现代的视窗系统大量使用了缓存来加快渲染文字的速度 - 甚至会把准备好的光栅保存到磁盘上，供多个应用使用，而且在系统重启后也不会丢失。然而，开发应用的程序员并不清楚他请求的字体是否已准备好并放入缓存，也不知道SetFontSize是否会进行（大量的）计算来获得字体中的所有字符，或是在字符串传递给DrawText时再按需要每次转换其中的一个字符。

功能更多的版本可能会允许程序员把字体数据保存在网络上的文件服务器中，所以读取字体可能会要点时间。如果服务器不响应，甚至还得长时间地等待超时，然后才会失败。

在很多情况下，细节无关紧要，但如果延迟很高或变化很大，那么应用可能会选择在自己的初始化时把拟使用的所有字体都准备好。大多数视窗系统的API未包含此类函数。

性能差异。“可预测”的函数性能可以通过其参数的属性进行估算。“无法预测”的函数则依赖于它们须执行的任务，差异颇大。打开存储设备的流所需的时间必定由访问时间决定，也可能还依赖于底层设备的数据传输速率。通过网络协议访问存储设备的开销可能非常昂贵；而且开销肯定不稳定。

很多低廉的函数只是在大多数时间如此，或者说人们对他们的期望开销较低廉。“读取字符（get a character）”的程序有时候必须通过调用操作系统重新填充缓冲区，这总是会比从满满的缓冲区内读取字符慢很多 - 而且有时候时间确实很长（例如，如果从一个负载较大的文件服务器读取文件，或是从快要崩溃的磁盘读取文件，只有反复多次读取后才能成功）。

由于各种不同的原因，性能“无法预测”的函数可能会表现出截然不同的性能。原因之一是函数蔓延（见侧边栏2）。例如，随着时间的变化，泛型函数的功能越来越越强。I/O流就是个佳例：调用打开流的方法时，会因流的类型不同而调用库和操作系统中截然不同的代码（本地磁盘的文件，网络服务器上的文件，管道，网络流，内存中的字符串等等）。随着I/O设备和文件类型的范围越来越大，性能的差异也只能随之增加。在多数API常见的生命周期中， 会随着时间的增加而增量地添加功能，性能的变化也会随之不可避免地增大。

另外，库在不同平台之间移植时的差异也是产生大量变化的原因。当然，平台-硬件和操作系统-的底层速度会有所不同，但是库的移植也可能导致API内部的函数出现*相对的*性能差异或者API之间的性能出现差异。人们往往会勉为其难，进行快速地移植，然后再慢慢解决移植带来的性能问题，这些已是常事。众所周知，有一些库在移植后性能变化悬殊，比如处理线程之类的库。线程异常可能以极端行为的方式表现 - 无比慢的应用，甚至死锁。

这些变化是构建精确的性能约定时面临的难题之一。通常没有必要高度准确地掌握性能，但若与期望的行为相差甚远，则会造成问题。

失败时的性能API的规范中详细描述了调用失败时的行为。常见的方式是返回错误代码和抛出异常，告诉调用者函数执行未能成功。然而， 正如规范中描述的正常行为一样，对失败时的*性能*也未做出任何说明。下文列举了三种重要的场景：

侧边栏 3.

如果打破了性能约定，会发生什么？

```
SetColor(g, COLOR_RED);
DrawLine(g,100,200,200,600);
```

图形库的函数通常较快。我们做个合理的假设，即你在绘制颜色各异的多条线条时，会设置每一条线的颜色；如果你担心SetColor函数不低廉，你可以只在颜色改变的时候调用。曾经在某个时候，出售工作站的公司提供了一个图形硬件产品，该产品要求在改变线条颜色前清除（硬件上的）所有几何流水线，这使得改变颜色的操作变得昂贵。举例来说，为了快起来，只得强行要求把线条按颜色排序，然后一次绘制所有的红色线条。这种方式与编写程序的直觉不符，并使得应用的结构和编程都出现了大的改动。

最后发觉昂贵的SetColor操作是一个错误，随后的硬件中也纠正了这个错误。但是，为克服这个错误而编写的代码可能仍然存在。对于不了解硬件性能变迁的代码维护人员来说，这成了一个错综复杂的问题，令人无法理解。

侧边栏 4.

关于malloc 和动态内存分配的经验之谈

如果把使用malloc()的动态内存分配归到“通常低廉”这一类，那也不错；但却有可能犯错，因为内存分配 - 特别是malloc - 已经成为性能问题的首要嫌疑对象之一。在探寻性能问题时，程序员首先怀疑的就是它。在得到了性能直觉方面的培养后，程序员收获很多，其中的一部分知识让他们懂得，如果他们调用了malloc几万次，特别是如果调用malloc来分配固定的小块内存，那倒不如用malloc分配一整块更大的内存，然后再管理他们自己的空闲区块列表。

malloc 的实现人员已经奋斗多年，试图让malloc在通常情况下忽视差异巨大的使用模式和运行时属性迥异的硬件/软件系统，在通常情况下保持低廉。[4] 但是，提供虚拟内存、线程和超大内存的众多系统均会对“低廉”造成挑战。malloc - 与它的配角free() - 必须在效率和某种使用模式的弊端（比如内存碎片）之间进行权衡。[8]

一些软件系统，比如Lisp 和Java，使用了自动的内存分配和垃圾回收来管理空闲的存储。虽然这极为方便，但是关注性能的程序员必须了解相关的开销。举例来说，应该一开始就告诉Java 程序员String对象和StringBuffer 对象之间的区别。修改String对象时，会在新的内存中创造新的副本，而StringBuffer对象则包含了空间来适应字符串变长的情况。随着垃圾回收系统越来越好，因垃圾回收而引发的无法预测的暂停发生得也越来越少；这可能会诱使程序员变得自满起来，相信内存的自动回收永远都不会产生性能问题，但是实际上性能问题只是比之前出现的少了一点而已。

‣ *迅速失败*。调用迅速返回失败 - 与正常的行为一样，或是更快。比如，调用 sqrt(-1) 可很快返回结果。即使由于内存不足，malloc 调用失败，返回结果需要的时间也应该与从操作系统中请求更多内存的其他malloc 调用相差无几。与成功的调用相比，调用时如果错误地打开流来读取不存在的磁盘文件，可能也不会花更多的时间。

‣ *缓慢失败*。有时候调用失败的时间非常长 - 时间太长了，应用程序也许会希望用其他的方式处理。例如，只有经历了几次长时间的超时之后，同另一台计算机建立网络连接的请求才可能会失败。

‣ *永远失败*。有时候，调用就是止步不前，完全不允许应用程序继续处理。举例来说，假如调用的实现需要等待同步锁，如果同步锁没有释放，那就永远不会返回。

与判断正常性能的直觉相比，判断失败性能的那种直觉完全不值一提。原因之一是在程序编写、调试以及调优时，程序员经历的失败事件比碰到的正常事件要少得多，这点容易理解。另一个原因是函数调用的失败可能有很多很多的方式，其中一些相当严重，而且API的规范中也未进行详尽地描述。即使在拟用于更精确地描述错误处理的异常机制中，也没有把所有可能的异常情况一一列出。不仅如此，随着库的功能越来越多，失败的可能性也越来越大。例如，封装网络服务（ODBC，JDBC，UPnP，……）的API本质上会承受大量繁杂的网络失败机制。

勤奋的应用开发人员会用大量的代码去处理不大可能发生的失败。常见的方式是把程序中的大块内容用try...catch语句包起来，这样便可重试失败的整个部分。交互性的程序则可以通过把整个程序放在巨大的try...catch中，试着保存用户的工作。通过把失败前用户已完成的工作结果记录在磁盘文件、关键日志或数据结构中，可以减轻主程序失败造成的后果。

处理停滞或死锁的唯一方法是设置一个监视线程，然后期望正常运行的应用程序定时在监视程序处签到，实际上就像在汇报“我现在仍然运行正常。”如果签到的间隔时间太长，监视程序就会采取行动 – 比如，保存状态，中止主线程和重启整个应用。如果交互式程序需要调用可能缓慢失败的函数来响应用户的命令，则可采用监视程序中止整个指令，返回到已知状态，这样便可允许用户继续执行其他命令。这促成了防御式的编程风格，为可能中止的每条命令做好准备。

为什么性能约定不容忽视？

为什么API必须遵守性能约定呢？*因为应用程序的主要结构可能依此类约定的遵守情况而定*。在选择API、数据结构以及总体的程序结构时，程序员在某种程度上会以API的性能期望为基础。如果期望或者性能出现严重偏差，那么程序员便无法仅仅通过调优API调用来进行弥补。相反，他必须重写大块的程序，乃至主要的部分。另一类例子则是上文中交互式程序的防御结构。

实际上，严重违反性能约定会导致综合失败：依照约定编写的程序无法匹配（组合）不遵守约定的实现。

当然，也有很多程序的结构和性能几乎不受库性能的影响（科学计算和大型仿真通常属于这一类）；不过，现在“常规IT”中的多数应用中大量使用了库，而且这些库的性能对总体性能起着决定性的影响。对于那些在基于Web的服务中广泛存在的软件而言，情况尤为如此。

性能的稍许波动甚至可能会让用户对程序的感知产生翻天覆地的变化。对于那些处理各种类型的媒体的程序来说，这点尤其正确。视频流中偶尔丢失几帧或许尚能接受（事实上，与其让帧速率滞后于其他媒体，丢失几帧更容易让人接受），但是人类的发展却让人可以发现音频中丢失的毫厘，所以此类媒体性能的细小变化可能会严重影响人们对整个程序的接受程度。这种担忧已经引起人们对服务质量产生浓厚的兴趣。在很多方面，这也是一种在高层级上确保性能的尝试。

如何避免违反约定的情况发生？虽然很少出现违反性能约定的情况，违反后造成灾难性的后果也很罕见，但在使用软件库时，关注性能的作用却可让软件更为健壮。下面列举了一些预防措施和策略供你参考：

*1. 仔细挑选API和程序结构*如果你可以享受从头开始编写程序这种奢侈，开始时便需要考虑性能约定引发的各种复杂情况。如果程序起初只是原型，然后会继续服务一段时间，那它无疑需要至少重写一次；重写也是你重新审视API和结构选型的良机。

2. 在新版本和移植发布时，API的实现者有义务提供前后一致的性能约定。即使是试验性质的新API也会引来用户使用，他们会开始构建API的性能模型。此后，如果改变了性能约定，无疑会激怒开发者，还可能导致他们重写程序。

API一旦成熟后，保持性能约定的稳定非常重要。事实上，最通用的API（例如，libc）之所以有通用的地位，可以说部分是由于在API演进的过程中，它们的性能约定一直稳定。对于API的移植来说，情况也是如此。

人们会希望，API实现人员可能会对新版本进行例行测试，验证他们没有引起性能的异常变化。不幸的是，几乎没有人做这类测试。但这并不是说你不能自己动手测试你依赖的API部分。利用探查器后，你通常可以发现程序只依赖少量的API。记录下老版本的性能，然后编写一套测试套件对库的新版

本和老版本进行性能比较，便可以让程序员得到预警，告诉他们在使用新的库版本后，他们自己的代码性能会发生变化。

很多程序员都-一致地-期望计算机和软件随着时间越来越快。也就是说，他们期望库或者计算机系统的每一个新版本都均匀地提升所有API的性能，特别是“低廉”的API。事实上供应商很难保证这点，但供应商却又把实际应用描述得非常贴近，使得客户都相信这点。很多工作站的客户都期望在使用图形库、驱动程序和硬件的新版本后，*所有*的图形应用的性能都能提升，但是他们同样关注各种功能上的提升，只是功能的提升通常会降低较老的函数的性能，即使只降了一点点。

人们还希望，API的规范中能清楚地描述性能约定，以便在使用、修改或移植代码时能够遵守约定。值得注意的是，如果函数使用了动态内存分配，无论是隐含的，或是自动的，都应该在文档中进行描述。

3. *利用防御式编程*。调用性能无法预测或性能变化巨大的API函数时，程序员可以采用特别的手段；考量失败时的性能时，这一点尤为正确。您可以把初始化从性能关键的区域中移出，并试着在使用API之前备好各种缓存数据（例如字体）。如果API性能差异巨大或者拥有大量的内部缓存数据，那么它可以提供一些函数让应用向API传递“暗示”，告诉API如何分配或初始化这些结构，这会有所帮助。通过间歇访问已确定需联系的服务器，可以建立可能无法连接的服务器清单，这样便可避免失败造成一些长时间的暂停。

在图形应用中有时候会用到这样的技术，就是做模拟运行，在屏幕之外（不可见）的窗口绘制一窗口的图形，这仅仅是为了准备好字体和一些图形数据结构的缓存。

很多程序员都-一致*地*-期望计算机和软件随着时间越来越快。

4. *调优API暴露的参数*。有些库提供了明确的方法来控制性能（例如，控制分配给文件的缓冲区的大小，表的初始大小，或者缓存的大小）。操作系统也提供了调优的选项。调整这些函数可以在性能约定的限制范围之内获取性能提升；调优虽然无法弥补严重的问题，但却可以用其他的手段减轻库中内置的固定选择对性能的严重影响。

有些库为函数提供了语义相同的替代实现方式，通常采取的形式为泛型API的具体实现。通过选择最佳的具体实现来进行调优往往易如反掌。Java 集合（Collections ）包就是这种结构的佳例。

在API的设计中加入适应运行中各种变化的因素，让程序员不再需要选择最佳的参数设置，这种情况越来越普遍。如果哈希表变得太满，会自动扩表和重新处理（唉，这种优点却被偶尔扩表带来的性能冲击抵消了）。如果顺序读取文件，便可分配更多的缓冲区，这样读的区块更大。

5. *通过度量性能来验证假设*。通常我们建议程序员监测关键的数据结构，以确定每个结构的使用是否正确。例如，你可以测量哈希表到底有多满，或者哈希冲突发生的频率高不高。或者，你还可以验证，在设计时如果一个结构为了读得更快而牺牲了写的性能，实际使用时这个结构是不是读比写多。

为了准确测量很多API调用的性能，需要添加足够的监控，这非常困难，也会涉及大量的工作，而且获得的信息也可能不值得这么做。然而，为那些决定应用性能的API调用添加监控（假定你已经确定了那些调用而且你的确定没错），便可在问题发生时节省大量的时间。我们还应该注意到，很多的这类代码还可以重用，作为下个库版本的性能监视的一部分，这点我们刚才已经谈过。

这些并不是想打击那些梦想家。他们想开发可以自动进行监视

和测量的工具，或是寻找规定性能约定的途径，以便证明性能度量符合约定。这些目标不容易实现，回报也可能没那么巨大。

一般情况下，即使不事先监测软件，也可以做性能度量（例如，利用探查器或类似DTrace的工具[5]）。这些工具的优点是，问题发生前什么都不用做，发生后追踪问题。当代码或者库的修改引发性能问题时，它们还可以帮你诊断偷偷混入的问题。正如《*编程珠玑*（*Programming Pearls*）》作者乔恩·本特利（Jon Bentley）所推崇的那样，“定期探查；利用信任的基线度量性能的偏差。”[1]

*6. 使用日志：发现和记录异常。*在由分布式服务组成的复杂系统领域，违反性能约定的情况越来越多。（请注意，通过网络接口提供的大多数服务有时候会带有SLA【服务等级协议】，其中规定了可接受的性能。在很多配置中，度量流程偶尔会发出服务请求来检验SLA是否满足。）因为人们使用了与API函数调用相似的方法通过网络连接调用这些服务（例如，远程过程调用或其变种，如XML-RPC、SOAP或 REST），所以性能约定中的期望也可以适用。应用会检测到这些服务的失败，做出的应变也恰到好处。然而，响应慢，尤其是数十个此类服务互相依赖时，系统性能可能会被迅速地击溃。专业管理的服务环境，比如那些提供Web服务的大型互联网站点，拥有精心开发的监测工具和相关工具监控Web服务的性能，并处理产生的问题。不过，家中小得多的计算机集群也依赖此类服务 - 很多成了各台笔记本电脑上的操作系统的一部分，有一些则内置在网络上的家用电器中（例如，打印机，网络文件系统，或文件备份服务） - 但却没有任何帮手帮你检测和处理性能问题。

如果这些服务的客户端能够记录他们期望的性能，并添加日志帮助人们诊断问题，那是再好不过了（syslog就是干这事的）。如果你的文件备份看起来出奇的慢（一小时备份200MB），想想，它比昨天更慢吗？与最近的操作系统更新之前的速度相比，还是更慢吗？考虑了几台计算机可能共享备份设备之后，还比你的期望值慢吗？或者，有没有合乎逻辑的解释（例如，备份系统发现了损坏的数据结构，开始了重建数据结构的漫长过程）？

诊断性能问题时，如果碰到了一团由不透明的软件（没有源代码，也缺乏组合中的模块和API的详细描述）组成的组合，则需要软件在报告性能和检测问题方面发挥作用。如果你不能从软件本身着手处理性能问题（它不透明），你只能对操作系统和网络进行调整和修补。如果备份设备因为磁盘快满了而变慢，那么你可肯定可以添加更多的磁盘空间。完备的日志和相关的工具能有点用；遗憾的是，在计算机系统的演进中，日志成了被低估和忽视的角落。

让组成发挥作用

如今，软件系统依托于各种独立开发的构件，按能够工作的方式把这些部件组合起来 - 也就是说，这些部件能以可接受的速度完成预期的计算。通过对组成进行静态检查来确保组成的正确（“通过软件组成验证正确性”），这仍是个遥不可及个梦想。与此相反，在软件工程实践中，发展了很多测试组件和组成的方法，这些方法都很管用。每当你把应用与动态库绑定时，或是在操作系统的接口上运行时，需要保证组成的正确性。

虽说组成的性能重要，但是在客户端和接口的提供方是否遵守彼此间的性能约定方面 - 人们并没有给予足够的重视。当然，这种约定不如正确性约定重要，但是发挥组成的全部能量又离不开它。

鸣谢

感谢巴特勒·兰普森（Butler Lampson），他创造了一个术语*低廉来描绘函数*，说明函数的性能之快让优化它们的所有努力都相形见绌；感谢埃里克·阿尔曼（Eric Allman），他提了一些有益的建议，说低廉带来了一个奇妙的机会，“经济但可用”；同时感谢乔恩·本特利（Jon Bentley），他是功底深厚的性能调试大师。

queue.acm.org 上的相关文章

《API设计不容忽视》*Michi Henning*
http://queue.acm.org/detail.cfm?id=1255422

《再论网络 I/O API：netmap框架》*Luigi Rizzo*
http://queue.acm.org/detail.cfm?id=2103536

《把语言穿过针眼》*Roberto Ierusalimschy, Luiz Henrique de Figueiredo and Waldemar Celes*
http://queue.acm.org/detail.cfm?id=1983083

参考资料

1. Bentley, J. Personal communication.
2. GNU C Library; http://www.gnu.org/software/libc/manual/html_node/index.html.
3. Java Platform, Standard Edition 7.API Specification; http://docs.oracle.com/javase/7/docs/api/index.html.
4. Korn, D.G., Vo, K.-P. In search of a better `malloc`.In *Proceedings of the Summer '85 Usenix Conference*, 489–506.
5. Oracle.Solaris Dynamic Tracing Guide; http://docs.oracle.com/cd/E19253-01/817-6223/.
6. Pthreads(7) manual page;http://www.kernel.org/doc/man-pages/online/pages/man7/pthreads.7.html; http://man7.org/linux/man-pages/man7/pthreads.7.html.
7. Saltzer, J.H., Kaashoek, M.F.Principle of least astonishment.In *Principles of Computer System Design*.Morgan Kaufmann, 2009, 85.
8. Vo, K.-P. Vmalloc: a general and efficient memory allocator.*Software Practice and Experience 26*, 3 (1996), 357–374; http://www2.research.att.com/~astopen/download/ref/vmalloc/vmalloc-spe.pdf.

罗伯特 F. 斯普劳尔（Robert F. Sproull）是马萨诸塞大学（阿默斯特）计算机科学系的兼职教授。从Sun 微系统实验室（Sun Microsystems Laboratories）主任的位置退下之后，他开始担任这一教职。

吉姆·沃尔多（Jim Waldo）是哈佛大学计算机科学实践方面的教授，兼任首席技术官。离开Sun 微系统实验室（Sun Microsystems Laboratories）之后，他开始担任这一教职。

译文责任编辑：崔斌

©2014ACM0001-0782/14/03$15.00

DOI:10.1145/2500875

稳定多线程大大简化并行程序的交叠行为，有望使并行编程更加容易。

作者：JUNFENG YANG、HEMING CUI、JINGYUE WU、YANG TANG 和 GANG HU

利用稳定多线程使并行程序变得可靠

可靠软件长期以来一直是大多数研究者、从业者和用户的梦想。在过去大约十年里，多项研究和工程学突破已经大大提高了串行程序（或并行程序的串行方面）的可靠性；成功的例子包括 Coverity 的源代码分析器、[6] Microsoft 的静态驱动程序验证程序、[3] Valgrind 内存检查器、[17] 以及验证的操作系统和编译器。[20]

但是，同样的成功并未惠及并行程序，并行程序有着难写、难测试、难分析、难调试和难验证的不良名声，其难度远远超过串行程序。专家将可靠并行视为“某种巫魔之术”[8] 并视其为巨大的计算难题之一。[1,18] 但是普遍的并行程序都面临潜藏的并发故障的困扰[15]（如数据竞争，程序并发访问同一内存位置，其中至少一次写入操作；以及死锁，线程循环等待资源）。其中某些最严重的情况造成了 Therac-25 放射治疗事件（因过量辐射而致人死亡）和 2003 年东北部大断电。攻击者可能利用这些程序故障来破坏关键系统的机密性、完整性和可用性。[24]

在过去十年里，两种技术趋势已使可靠并行的挑战变得更为紧迫：第一种趋势是多核硬件的兴起；单个处理器内核的速度受限于底层物理限制，这迫使处理器采取多核设计，开发者依赖并行代码在多核处理器上获得最佳性能。第二种趋势是计算需求增长加快。科学计算、视频和图像处理、金融模拟、大数据分析、Web 搜索以及在线社交网络，这些全都需要海量计算和各种并行程序来最大化性能。

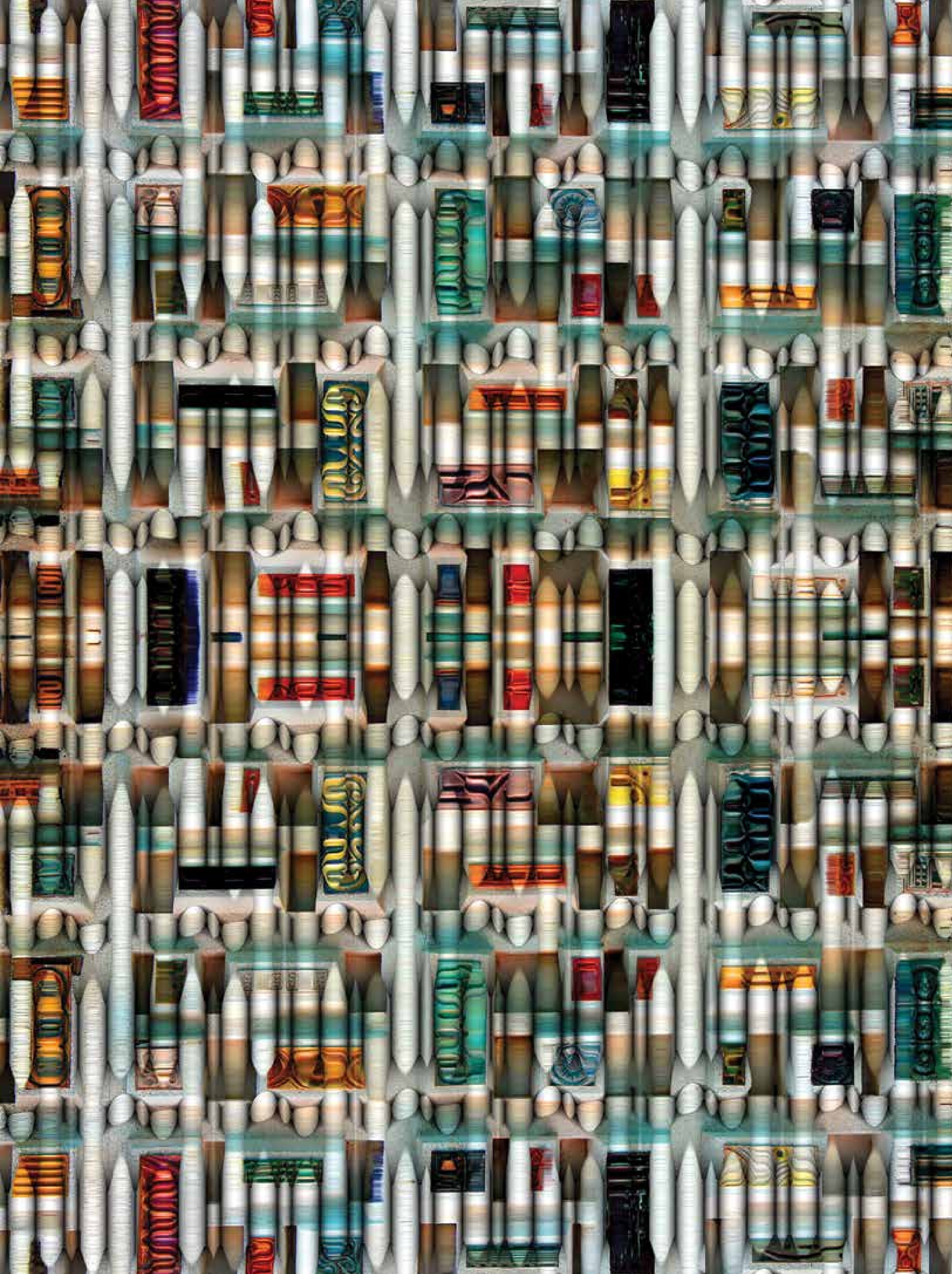
如果说可靠软件是横跨整个计算机科学的总体难题，那么可靠并行则无疑是关键。为了让并行程序变得可靠，研究人员投入数十年努力，产生了很多想法和系统 — 从新的硬件、新的编程语言和新的编程模型，到可检测、诊断、避免或修复并发故障的各种工具。新的硬件、语言和模型即使得到采用，通常也需要数年时间。工具虽然很有帮助，但是往往只解决衍生出的问题，而不是解决根本原因。

我们希望解决让共享内存的多线程程序可靠的悬而未决的根本性问题。这些程序通过线程来表达并发性，线程是一种共享内存的轻量级顺序进程。我们讨论这类程序是因为它们是最普遍的并行程序类型，有着硬件、操作系统、库和编程语言的成熟支持。而且，在可预见的将来，它们很可能依然是主流。

» 重要见解

- **要让多线程程序不出错很难，因为它们有太多可能的线程交错，或者说调度方案，而不是因为它们是非确定性的。**
- **并非所有调度方案都是必需的；对很多程序而言，少量调度方案就足以处理所有可能的输入。**
- **StableMT 大幅减少调度方案，同时将开销压得很低，从而大大增强几乎所有可靠性技术，包括测试、调试和程序分析。**

JOEL ORMSBY



与顺序程序不同的是，对同一输入反复执行同一多线程程序可能产生不同的行为（如正确的行为或错误的行为），具体取决于线程如何交叠。传统观点一直认为这种不确定性是造成这些可靠多线程难题的罪魁祸首；[13]线程默认就是非确定性的，考虑这种非确定性是开发人员的（棘手）任务。非确定性对可靠性有直接影响；例如，它使得测试效果更差。程序在测试实验室中可能对某个输入正确运行，因为测试的线程交叠碰巧正确；但是在现场，在程序遇到未经测试的错误交叠时，程序对完全一样的输入仍可能执行失败。

为了消除不确定性，包括我们在内的多个研究小组致力于构建确定性多线程（DMT）系统，[2,4,5,7,12,14,19]，这类系统强迫多线程程序对同一输入总是执行同样的线程交叠（或称调度方案），从而总是产生相同的行为。通过将每个输入只映射到一个调度方案，DMT 将确定性（串行计算的关键属性）引入了多线程。

但是，非确定性只是难题的一小部分，而确定性（治疗非确定性的灵药良方）也并非通常想像得那样有用，对可靠性来说，它既不充分，也非必要。说它不充分是因为，完全确定性的系统也可以将每个输入映射到任意调度方案，因此很小的输入扰动会导致相差极大的调度方案，从而人为降低程序的牢固性和稳定性。说它非必要是因为，如果某个非确定性系统的所有输入只有很少的调度方案，那么通过全面检查所有调度方案，也可以使系统变得可靠。

造成程序员难以正确处理多线程的原因是数量；多线程程序的调度方案过多。根据硬件计时和操作系统调度等因素，并行线程可按多种方式交叠，因此每个输入的调度方案数已然很大。汇总所有输入后，这个数字还要大。从所有调度方案中找出触发并发错误的调度方案（这样开发人员就可防止这类错误）就像是大海捞针。虽然 DMT 减少了每个输入的调度方案，但是它可能将每个输入映射到另一个不同的调度方案，因此所有输入的总调度方案数依然巨大。

从“海底”去掉不必要的调度方案会使“捞针”更容易。

我们解决这一根本原因的方法是，判断所有这些调度方案是否都是必要的。我们 2010 年的研究发现，很多现实程序只使用少量调度方案即可高效处理一系列输入。[10]利用这一发现，我们构想了一种新的方法，我们称之为稳定多线程，或称 StableMT，这种方法对一系列输入重用每一个调度方案，将所有输入映射到数量已大大减少的一组调度方案。通过大大缩小“海”的范围，捞“针”就会变得容易得多。将很多输入映射到同一调度方案，程序行为可变得稳定，可抗拒少量输入扰动。StableMT 和 DMT 并不互斥；一个系统既可以是确定性的，又可以是稳定的。

为了实现我们的 StableMT 愿景，我们构建了多个系统：TERN[10] 和 PEREGRINE；[11]StableMT 的两种编译器和运行时实现；以及一个程序分析框架，此框架可利用 StableMT 达到其对手框架无法企及的高覆盖率和精度。[22]它们解决了三个互补的难题，其中的两个长期以来在相关领域中一直未得到解决。TERN 解决如何计算可高度重用的调度方案。调度方案越可重用，需要的调度方案就越少。遗憾的是，计算可重用调度方案在编译时无法确定，在运行时代价高昂。PEREGRINE 解决如何有效地使执行过程遵循调度方案，而不偏离；在确定性执行和重放领域，这是一个持续了数十年的老大难问题。我们的分析框架解决如何有效分析多线程程序，这在程序分析领域是众所周知的未解难题。这些系统的实现对开发人员几乎透明，并完全兼容现有硬件、操作系统、线程库和编程语言，从而方便大家采用。

我们的初步成果非常成功。对各种多线程程序（包括 Apache Web 服务器和 MySQL 数据库）的评估表明 TERN 和 PEREGRINE 大大减少了调度方案的数量；例如，在典型设置下，不管文件内容是什么，它们将并行压缩实用工具 PBZip2 所需的调度方案数量减少到每种不同的线程数只有两个调度方案。它们的开销对大多数程序来说中等，不到 15%。我们的程序分析框架能够构建精度和覆盖率令其他同类框架难以望其项背的很多程序分析工具；例如，我们构建的数据竞争检测器在经过全面检查的代码中发现了以前未知的程序故障，而且几乎没有误报。

很难做对

下面，我们从最基础的部分开始，描述由非确定性以及过多调度方案造成的难题，并解释为什么非确定性与过多调度方案相比，只能算次要原因。

输入、调度方案和有故障调度方案。我们说的“输入”广义上是指程序从其执行环境读取的数据，不仅包括从文件和套接字读取的数据，还包括命令行参数、外部函数的返回值（如 gettimeofday）以及可能影响程序执行的任何外部数据。我们说的“调度方案”广义上是指多线程执行过程中通信操作的全序或偏序集合，包括同步（如 lock 和 unlock 操作）和共享内存访问（如对共享内存的 load 和 store 指令）。在所有调度方案中，大多数都运行良好，但是有一些会触发并发错误，造成程序崩溃、不正确的计算、死锁以及其他故障。请考虑下面这段小程序：

```
// thread        // thread 2
lock(l);         lock(l);
*p = . . .;      p = NULL;
unlock(l);       unlock(l);
```

线程 2 在线程 1 前获得锁的调度方案会引起 dereference-of-NULL 错误。现在考虑另一个示例，这个示例对 balance（余额）有数据竞争：

```
// thread 1        // thread 2
// deposit 100     // withdraw 100
t = balance + 100;
                   balance =
                   balance - 100;
balance = t;
```

语句按此顺序执行的调度方案将破坏余额。我们将这种触发并发错误的调度方案称为“有故障调度方案”。严格来说，这些错误存在于程序中，由输入和调度方案共同触发。但是，典型的并发错误（如 Lu 等人[15] 和 Yang 等人[24] 的文章所述）更多取决于调度方案，而非输入（例如，一旦调度方案固定，该调度方案允许的所有输入都出现故障）。对多线程程序测试的近期研究（如 Musuvathi 等人[16]）侧重于测试调度方案来找出有故障调度方案。

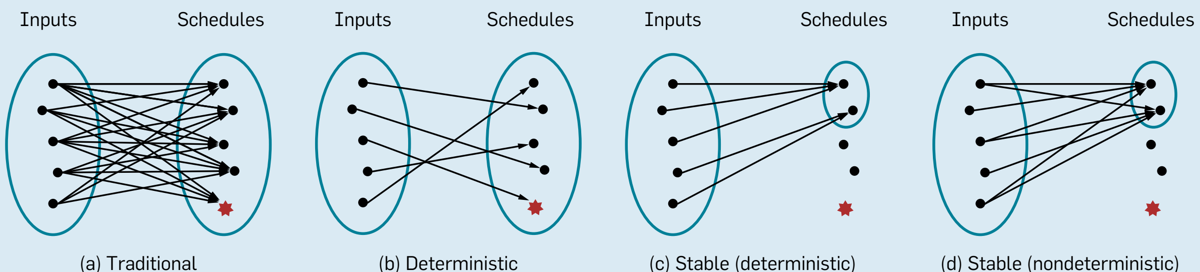
非确定性造成的难题。多线程程序是非确定性的，因为即使同一个程序和输入，不同的调度方案仍可能导致不同的行为；例如，上面两段小程序并不总是会发生故障。除了上面所述的调度方案外，其他调度方案会产生正确的执行。

这种非确定性带来很多难题，尤其是在测试和调试方面。假设一个输入可按 n 个调度方案执行。测试 $n-1$ 个调度方案不足以保证完全可靠性，因为一个未测试的调度方案仍可能有瑕疵。现场执行可能遇到这个未经测试的调度方案并失败。调试也很艰难。为了重现现场故障以供诊断，单单一模一样的输入还不够；开发人员还必须设法从 n 种可能性中重构有瑕疵的调度方案。

图 1a 概略描绘了传统多线程方法，从概念上说，这是一种多对多映射，其中一个输入可按多个调度方案执行（因为不确定性），而多个输入也可只按一个调度方案执

图 1.多线程方法。

红星表示有瑕调度方案。传统多线程 (a) 从概念上说是一种多对多映射，其中一个输入可按多个调度方案执行（因为不确定性），而多个输入也可只按一个调度方案执行，因为调度方案修正了通信操作的顺序，但允许本地计算操作任何输入数据。DMT (b) 可将每个输入映射到调度方案，从而降低程序抗输入扰牢固性动的牢固性。StableMT（c 和 d）减少了所有输入的全部调度方案（由缩小的椭圆表示），从而提高牢固性和可靠性。StableMT 和 DMT 是正交的；StableMT 系统可以是确定性 (c) 的，也可以是非确定性 (d) 的。



行，因为调度方案修正了通信操作的顺序，但允许本地计算操作任何输入数据。

过多调度方案造成的难题。典型的多线程程序包含非常大量的调度方案。对单个输入来说，调度方案数接近调度方案长度的指数；例如，给定 m 个线程，每个线程争用一把锁 k 次，每一个获取锁的顺序构成一个调度方案，很容易得出 $\frac{(mk)!}{(k!)^m} \geq (m!)^k$ 个总调度方案数，这个数字是 m 和 k 的指数。汇总所有输入后，调度方案数还要更大。

在如此之多的调度方案中，只查找几个有瑕调度方案会引发一系列海底捞针难题；例如，为了编写正确的多线程程序，开发人员必须小心同步其代码，以剔除有故障调度方案。而当必须全面检查很多种可能性来查找极个别案例时，人类通常难免犯错。各种形式的测试工具也面临同样的问题。压力测试是（间接）测试调度方案的常用方法，但通常只是冗余地测试相同的调度方案，而非遗漏的调度方案。最近的工具（如 Musuvathi 等人开发的工具[16]）系统性地测试调度方案来查找程序故障，但开发人员缺乏资源，只能覆盖数量庞大的所有调度方案中的极小一部分。

确定性不如通常想像得那样有用。为了克服非确定性带来的难题，包括我们在内的研究者投入了大量精力，并构建出多种系统，这些系统强制多线程程序总是对同样的输入运行相同的调度方案，从而将确定性引入多线程。确定性对可靠性是有价值的；例如，一次测试执行现在可验证对同一输入的所有未来执行，而重现并发错误现在只需要提供相同输入即可。

另一方面，很少有工作解决过多调度方案造成的难题。研究界过多归咎于非确定性，而忽略了主要原因 — 一个相当量化的原因，即多线程程序的调度方案过多。虽然确定性很重要，但其重要性并不如通常认为的那样大，对可靠性而言，它既不充分，也非必要。

确定性 $\not\Rightarrow$ 可靠性。确定性是刻板的属性：同样的输入 + 同样的程序 = 同样的行为。如果输入或程序发生变动，无论变动多么轻微，它都无权控制。但是开发人员通常希望程序在程序或输入发生轻微变动的时候保持鲁棒和稳定；例如，添加一句调试 printf 原则上不应使程序故障消失。同样，文件只变动一位 (bit) 通常不应造成压缩实用工具崩溃。遗憾的是，确定性未提供这样的稳定性。简单的确定性实现甚至可能削弱稳定性。

为了说明这一点，请考虑图 1b 中的系统，该系统将每一个输入映射到一个任意调度方案。此映射是完全确定性的，但在多个输入时会造成程序行为不稳定。变动一位可能迫使程序放弃正确的调度方案，并随意选择截然不同的有故障调度方案。

这种不稳定性至少有违直觉，从而引发新的可靠性难题；例如，测试一个输入并不能保证非常相似的输入也没有问题，哪怕输入中的差异并未导致被测试的调度方案无效。调试现在需要所有引入故障的输入，不仅包括用户输入的数据，还包括环境变量、共享库，甚至不同的用户名，或者错误报告是否缺少信用卡号。不管开发人员重试多少次，程序故障可能永不重现，因为确定性系统为变动过的输入选择的调度方案碰巧正确。请注意，即使正确的串行程序在碰到跨边界条件的输入变化时，也可能表现出不同的行为，但是这类情况通常很少见，而且不同的行为是开发人员故意为之。相反，图 1b 中的系统引入的不稳定性是人为的，并所有输入都会引起这种不稳定。

除了输入外，简单实现的确定性可能只要代码稍稍更改，就会使程序行为不稳定，因此添加一句调试 printf 可能会使程序故障消失。另一个问题是，所有可能的调度方案的数量依然巨大，因此调度方案测试工具的覆盖率仍然很低。

实践中，为了缓解上述问题，研究人员采用其他技术作为确定性的补充。为了支持调试 printf，有人提出暂时恢复为非确定性执行。[12]此外，DMP[12] 以及 CoreDet[4] 和 Kendo[19] 只在输入更改所执行的底层指令时，才更改调度方案。虽然这些系统要比将每个输入映射到任意调度方案来得好，但在我们的实验中观察到，当输入扰动更改了执行的底层指令（如执行一次额外的 load）时，它们仍能让少量输入扰动不必要地破坏调度方案稳定性。[10]我们的 TERN 和 PEREGRINE 系统以及 Liu 等人在 TERN 之后构建的 DTHREADS[14] 将 DMT 与 StableMT 结合在一起，对一系列输入频繁重用调度方案，以获得稳定性。

可靠性 $\not\Rightarrow$ 确定性。确定性是二元属性；如果一个输入映射到 $n > 1$ 个调度方案，那么不管 n 有多小，对此输入的执行仍可能是非确定性的。但是很容易使总调度方案数很小的非确定性系统变得稳定。请考虑图 1d 中非确定性系统的极端案例，它将所有输入映射到最多两个调度方案。此案例中，开发人员可以很容易解决非确定性引起的海底捞针难题；例如，为了重现给定输入引起的现场故障，他们可以承受搜索唯一两个调度方案的其中之一。打个比方，抛硬币是非确定性的，但人类对于理解和玩耍抛硬币没有问题，因为只有两种可能的结果。

缩小“海底”范围

在确定性的局限性以及过多调度方案所引发的难题的驱使下，我们考察了一个核心研究问题：是否所有数量如此庞大的调度方案都是必需的。如果调度方案在特定场景下是唯一可处理特定输入或产生良好性能的调度方案，则该调度方案是必

需的。从“海底”去掉不必要的调度方案会使“捞针”更容易。

我们调查了各种流行的多线程程序，从服务器程序（如 Apache）到桌面实用工具（如并行压缩实用工具 PBZip2），再到计算密集型算法的并行实现（如快速傅立叶转换）。它们使用各种同步原语（如锁、信号量、条件变量和栅）。这产生了两条见解：第一，对很多程序来说，由很多输入组成的一系列输入共享同一等效类的调度方案。该类中的一个调度方案就足以处理整个输入系列。直觉上，一个输入通常包含两种类型的数据：控制执行过程的通信的元数据（如要派生的线程数），以及线程在本地计算的计算数据。某个调度方案要求输入的元数据具有特定值，但也允许计算数据发生变动。也就是说，它可处理有着相同元数据的任何输入；例如，请考虑 PBZip2，它将输入文件在多个线程之间拆分，每个线程压缩一个文件块。其通信（即哪个线程获得哪个文件块）与线程本地压缩无关。在典型设置下（如无读取失败或信号），对于用户设置的每一种不同的线程数，PBZip2 可能使用两个调度方案。不管文件数据内容是什么，如果文件可以均匀地按线程数划分，则使用一个调度方案，否则使用另一个调度方案。

输入与调度方案间的这种松散耦合并不是 PBZip2 独有的；很多其他程序也展现出这种属性。表 1 是我们的发现结果的一个示例，其中包括三个现实程序 — Apache、PBZip2 和 aget（并行文件下载实用工具）— 以及计算密集型算法的五种实现，这些实现来自两个广泛使用的基准程序套件 — Stanford 的 SPLASH2 和 Princeton 的 PARSEC。

第二条见解是，对不同输入实施调度方案的开销很低。人们很可能认为，数量呈指数级放大的调度方案很可能允许运行时系统对各种计时因素作出反应，并选择有效的调度方案。但是，我们 StableMT 系统的结果却推翻了这种假设。对大多数受评估的程序来说，利用设计谨慎的调度方案表示形式，在对不同的输入实施调度方案时，它们引起的开销不到 15%。这种中度的开销相比获得的可靠性还是值得付出的。

利用我们的见解，我们发明了稳定多线程，或称 StableMT，一种全新的多线程方法，这种方法对一系列输入重用每一个调度方案，将所有输入映射到数量已大为缩小的一组调度方案。一次性大幅缩小“海底”。此外，对映射到同一调度方案的输入以及不影响调度方案的轻微程序更改，StableMT 可使程序行为稳定，从而提供开发人员和用户等群体所期望的鲁棒性和稳定性。

StableMT 和 DMT 是正交的。StableMT 旨在减少所有输入的调度方案，而 DMT 旨在减少每个输入的调度方案（减少到一）。StableMT 系统可以是确定性，也可以是非确定性。图 1c 和图 1d 描绘了两个 StableMT 系统；图 1c 中的多对一映射是确定性的，而图 1d 中的多对少映射是非确定性的。多对少映射提高了性能，因为运行时系统可根据当前计时因素，从少数几个调度方案中为输入选择一个高效的调度方案，但是增加了为获得可靠性所需的工作和资源。幸运的是，只有少数几个调度方案可选择（比如一个很小的常数，如 2），因此由非确定性造成的难题很容易解决。

益处。通过大幅减少调度方案，StableMT 为多线程带来了很多可靠性益处：

*测试。*StableMT 自动提高调度方案测试工具的覆盖率，覆盖率定义为受测试的调度方案与全部调度方案之比；例如，请再次考虑 PBZip2，在典型设置下，它对每一个不同的线程数，只需要两个调度方案。测试 32 个调度方案覆盖 1 到 16 个线程。考虑到当线程数等于或接近内核数时，PBZip2 将达到峰值性能，并且典型的计算机最多有 16 个内核，因此 32 个受测试的调度方案可以覆盖现场执行的大多数调度方案。

*调试。*重现程序故障不需要完全一样的输入，只要原来的输入和更动后的输入映射到同一个调度方案即可。也不需要完全一样的程序，只要程序的变动不影响调度方案即可。用户可以从其程序故障报告中去除隐私信息（如信用卡

表 1.共享同一等价类调度方案的输入的约束。对于每个程序，在典型设置下（如无系统调用失败或信号），该类中的一个调度方案就足以处理满足第三列中约束的任何输入。

程序	用途	共享调度方案的输入的约束
Apache	Web 服务器	对于一组典型的 HTTP GET 请求，缓存状态相同
PBZip2	压缩	线程数相同
aget	文件下载	线程数相同，文件大小相似
barnes	N 体模拟	线程数相同，两个配置变量的值相同
fft	快速傅立叶转换	线程数相同
lu-contig	矩阵分解	线程数相同，矩阵和块大小相似
blackscholes	期权定价	线程数相同，期权数不小于线程数
swaptions	掉期期权定价	线程数相同，掉期期权数不小于线程数

号），而开发人员可以在不同的环境中重现故障或添加 printf 语句。

*分析和验证程序。*静态分析可以集中于那些在现场可能实施的调度方案集以获得精度。动态分析则获得跟测试一样的益处。模型检查可检查明显减少的调度方案，从而缓解所谓的“状态爆炸”问题。[9]交互式定理证明也变得更加容易，因为验证者只要对现场实施的调度方案集证明定理。

*运行时避免错误。*程序也可在现场自适应地学会正确的调度方案，然后对未来的输入重用它们，以避免未知的、可能有瑕疵的调度方案。

告诫。StableMT 并非适合每一个多线程程序。它对调度方案与输入松散耦合的程序很有效，但是程序可能根据涉及输入中多个位的复杂情况来决定采取某种措施，例如派生线程或调用同步。例如，类似 grep 的并行实用工具 pfscan，它使用多线程在一组文件中搜索某个关键字，并为每个匹配项获取一个锁来递增计数器。对一组文件计算的调度方案不可能适合另一组文件。若要提高每一个调度方案覆盖的输入范围，开发人员可使用批注 (annotation) 从调度方案上排除对此锁的操作。

当输入和程序的轻微扰动不影响调度方案时，StableMT 提供了可抗这些扰动的鲁棒性和稳定性，尽管仍有改进空间；例如，当开发人员通过添加同步来更改其程序时，更新以前计算的调度方案要比从头重新计算更高效。我们将此想法留给以后的工作。

构建稳定的多线程系统

虽然稳定多线程的前景非常诱人，但其实现面临诸多挑战，包括：

*计算调度方案。*StableMT 系统如何计算输入要映射到哪个调度方案？调度方案在现实情况中必须可行，这样重用它们的执行过程才不会“卡壳”。它们还可应该高度可重用；

*实施调度方案。*StableMT 系统如何确定性并高效地实施调度方案？只有做到“确定性”，重用调度方案的执行过程才不会偏离这个调度方案，即使在存在数据竞争的情况下也要达到调度方案的确定性；而只有“高效”，这样开销才不会抵消可靠性收益，在确定性执行和重放领域，这个难题数十年未解；

*处理线程。*StableMT 系统如何处理多线程服务器程序？这类程序通常长时间运行，它们在客户端请求到来时，对每一个请求作出反应，因而使其调度方案非常特定于请求流，并难以重用。

从 2009 年起，我们就一直在克服这些挑战；我们构建了两个 StableMT 原型 — TERN[10] 和 PEREGRINE[11]，它们以很低的开销频繁重用调度方案。这里我们将说明我们的解决方案，尽管它们绝不是唯一的解决方案；例如，继 TERN 之后，其他研究人员也构建了一套系统，此系统可以稳定常规多线程程序的调度方案。[14]

计算调度方案。对于实现 StableMT 至关重要的一点是，如何计算用于处理输入的调度方案集合。最低限度，在对输入实施调度方案时，调度方案必须可行，这样执行过程才不会“卡壳”或偏离调度方案。理想状态下，考虑到可靠性，调度方案的集合还应该小。一种想法是，使用静态源代码分析来预先计算调度方案，但是停机问题使得静态计算保证可动态工作的调度方案是不可判定的。另一种想法是在程序运行时动态计算调度方案，但是计算可能非常复杂，其开销可能高得难以承受。

图 2.基于并行压缩实用工具 PBZip2 的示例程序，它派生出 nthread 个工作线程，在线程间拆分文件，然后并行压缩文件块。

```
1:  main(int argc, char *argv[ ]) {
2:    int i, nthread = atoi(argv[1]);
3:    for(i=0; i<nthread; ++i)
4:      pthread_create(worker); // 创建工作线程
5:    for(i=0; i<nthread; ++i)
6:      worklist.add(read_block(i)); // 将块添加到工作列表
7:    // 错误：缺少 pthread_join() 操作
8:    worklist.clear(); // 清空工作列表
9:    …
10: }
11: worker() { // 压缩文件块的工作线程
12:   block = worklist.get(); // 从工作列表获取文件块
13:   compress(block);
14: }
15: compress(block t block) {
16:   if(block.data[0] == block.data[1])
17:     …
18: }
```

图 3.示例程序的同步调度方案。每个同步标有其在图 2 中的行号。

```
// main                    // worker 1           // worker 2
4: pthread_create(worker);
4: pthread_create(worker);
6: worklist.add();
                           12: worklist.get();
6: worklist.add();
                                                 12: worklist.get();
8: worklist.clear();
```

我们的系统通过记录过去执行过程中的调度方案来计算调度方案；记录的调度方案然后可重用于以后的输入，以稳定程序行为。TERN 的工作方式如下：在运行时，它保留一个持久缓存，其中保存从以前的执行过程中记录的调度方案。当输入到达时，它会搜索缓存，寻找与该输入兼容的调度方案。如果找到一个兼容的调度方案，它直接运行程序，同时实施该调度方案。否则，它原样运行程序，同时记录执行过程中的新调度方案，将新调度方案存入缓存以供将来重用。

计算调度方案的 TERN 方法有几点好处：首先，通过重用表现正常的调度方案，可以避免未知调度方案中的潜在错误，从而提高可靠性。现实世界中类似的例子是，自然人类（和动物）倾向于走熟悉的路线，以避免未知路线可能出现的危险。例如，候鸟通常遵循固定的飞行线路。（TERN 这个名字源自于 Arctic Tern，即北极燕鸥，一种在所有动物中迁徙最远的鸟类。）为什么我们的多线程系统不能向它们学习，并重用熟悉的调度方案？

其次，TERN 显式存储调度方案，这样开发人员和用户就可以灵活选择要记录哪些调度方案以及何时记录；例如，开发人员可以在测试期间填充正确调度方案的缓存，然后将此缓存与其程序部署在一起，从而提高测试效率，并避免用户计算机上记录调度方案的开销。而且，他们可以对调度方案运行自己喜欢的检查工具，以检测各种错误，并选择在缓存中只保留正确的调度方案。

TERN 之所以高效是因为，它可以摊销计算调度方案的成本。记录并检查调度方案比重用调度方案开销更大，但幸运的是，TERN 对每个调度方案只记录并检查一次，然后对多个输入重用调度方案，从而摊薄开销。

TERN 的关键难题是在按照调度方案执行输入之前，检查输入是否与调度方案兼容。如果不兼容，那么在它尝试实施不兼容的调度方案时（例如对需要四线程的输入实施两线程的调度方案），执行过程将不会遵循调度方案。这个难题变成了我们在构建 TERN 的过程中必须解决的最难难题。我们的解决方案利用多种高级程序分析技术，包括我们发明的两项新技术；详情请见 Cui[10] 和 Cui[11]。

在记录调度方案时，TERN 跟踪调度方案中的同步与输入间的依赖关系，将这些依赖关系捕获到一个松散的、可快速检查的约束集中，我们将此约束集称为“调度方案的先决条件”。然后，它将该调度方案重用于满足先决条件的所有输入，从而避免重新计算调度方案的运行时开销。

计算先决条件的朴素方法是，从执行过程中所有跟输入相关的分支中收集约束；例如，如果某个分支指令检测输入变量 x，然后转向 true 分支，TERN 就会将 x 必须非零的约束添加到先决条件。这样计算的先决条件虽然充分，但包含很多只与线程本地计算相关的非必要约束。由于过分约束的先决条件会降低调度方案重用率，因此 TERN 从先决条件中去除这些不必要的约束。

我们演示一下 TERN 如何处理某个简单程序，这个程序基于前面提到过的并行压缩实用工具 PBZip2（请见图 2）。其输入包括 argv 中的所有命令行参数以及输入文件数据。为了压缩文件，它派生出 nthread 个工作线程，相应地拆分文件，然后调用函数 compress 来并行压缩文件块。为了协调工作线程，它使用同步的工作列表。（这里为清楚起见，我们使用工作列表同步；实际上，它处理 Pthread 同步。）但是，这个示例有个程序故障：因为第 7 行缺少 pthread_join 操作，因此在第 8 行清除工作列表后，worker 函数可能仍使用此工作列表，从而造成潜在的程序崩溃；此个程序故障是基于 PBZip2 中的真实故障。

我们先说明 TERN 如何记录调度方案及其先决条件。假设我们用两个线程运行此示例，并且 TERN 记录了可避免“释放后使用”故障的调度方案（请见图 3）。（其他调度方案也是可能的。）为了计算该调度方案的先决条件，TERN 先记录依赖于输入数据的所有已执行分支语句的结果；图 4 包含所收集的约束的集合。然后，它应用高级程序分析来去除只与本地计算有关，且对调度方案无影响的那些约束，包括从函数 compress 收集的所有约束。余下的约束简化为 nthread = 2，形成该调度方案的先决条件。TERN 将调度方案和先决条件存入调度方案缓存。

现在我们说明 TERN 如何重用调度方案。假设用户也想用两个线程来压缩一个完全不同的文件。TERN 会检测到 nthread 满足 nthread = 2，因此它重用图 3 中的调度方案来压缩文件，而不考虑文件的数据。此执行是可靠的，因为调度方案避免了“释放后使

图 4.为该调度方案收集的所有输入约束，每个约束标有其在图 2 中的行号。这些从压缩函数收集的约束以后将由 TERN 移除，因为它们对调度方案没有影响；余下的约束简化为 nthread = 2。

```
3:0 < nthread ? true
3:1 < nthread ? true
3:2 < nthread ? false
5:0 < nthread ? true
5:1 < nthread ? true
5:2 < nthread ? false
16: … // 从 compress() 收集的约束
```

用”故障。它还非常高效，因为调度方案只对同步操作进行排序，并允许压缩操作并行运行。假设用户再次用四个线程运行此程序。Tern 会检测到输入不满足先决条件 nthread = 2，因此将记录新的调度方案和先决条件。

高效实施调度方案。过去的工作以两个不同的粒度（共享内存访问或同步）实施调度方案，迫使用户在效率和确定性之间折衷。具体来说，内存访问调度方案将使数据竞争具有确定性，但是严重低效（例如，要比传统多线程慢 1.2 到 6 倍[4]）；同步调度方案要高效得多（例如，平均只下降 16%[19]），因为它们是粗粒度的，但是不能令有数据竞争的程序具有确定性（如前面讨论的第二个小程序，以及很多现实多线程程序[15,23]）。这一确定性和性能的之间的矛盾难题在确定性执行和重放领域数十年未解决。因此，我们的第一个 StableMT 系统 Tern 只实施同步调度方案。

正是这个难题促使我们构建第二个 StableMT 系统 Peregrine[11]。Peregrine 的设计理念是，虽然很多程序都有数据竞争，但是数据竞争倾向于只在执行过程的一小部分中发生，执行过程的大部分仍然是无竞争的。直觉上，如果某个程序到处是数据竞争，那么大多数数据竞争应已在测试期间被捕获。我们实际分析了七个有数据竞争的真实程序的执行过程，最终发现尽管有成千上万的内存访问，但每个执行过程只有最多 10 个数据争用。

由于争用非常少，因此对于执行过程的无争用部分，Peregrine 可以调度同步，而只对“有争用”的部分诉诸于调度内存访问，从而将同步调度方案的效率与内存访问调度方案的确定性结合在一起。这些混合调度方案的粒度几乎与同步调度方案一样粗，因此也可以频繁重用（请见图 5）。

Peregrine 如何在执行实际开始前预测哪里可能发生数据竞争？一种想法是，使用静态分析在编译时检测争用。但是，静态数据竞争检测器素有欠准确的不良名声，因为其大部分报告往往是误报，而非真正的数据竞争。调度误报中的大量内存访问将严重减慢执行过程。Peregrine 利用 Tern 中的“记录再重用”方法预测数据竞争；所记录的执行过程可有效预测重用同一调度方案的执行过程可能发生什么情况。具体来说，在记录同步调度方案时，Peregrine 记录详细的内存访问记录。Peregrine 从内存访问记录中检测实际发生的数据竞争（与调度方案有关），将涉及竞争的内存访问添加到调度方案。这种混合调度方案可高效且确定性地实施，从而解决前面提到的一直悬而未决的“确定性对性能”难题。为了将该调度方案重用于其他输入，Peregrine 提供了新的先决条件计算算法，以保证重用该调度方案的执行过程不会遇到任何新的数据竞争。为对实施内存的按序访问，Peregrine 通过一个安全高效的程序插桩框架在运行时修改正在运行的程序，这个框架名为 Loom，是我们在 2010 年构建的。[21]

处理服务器程序。服务器程序向 StableMT 提出了三大难题：第一，它们可能连续运行，造成其调度方案数量实际无穷无尽，而且过于特定而难以重用；其次，它们通常在请求到达时就立即处理输入

图 5.混合调度方案。

圆圈表示同步，三角表示内存访问。同步调度方案非常高效，因为它是粗粒度的，但非确定性的，因为数据争用仍可能造成执行过程偏离调度方案并失败。内存访问调度方案是确定性的，但是很慢，因为它是细粒度的。混合调度方案兼具两者的最佳，它只为执行过程的争用部分调度内存访问，而其他部分则调度同步。

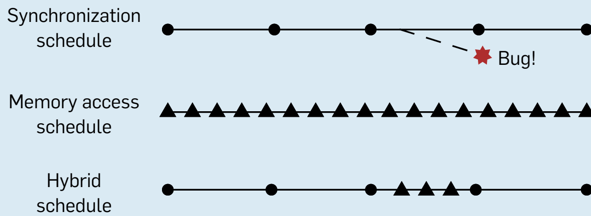
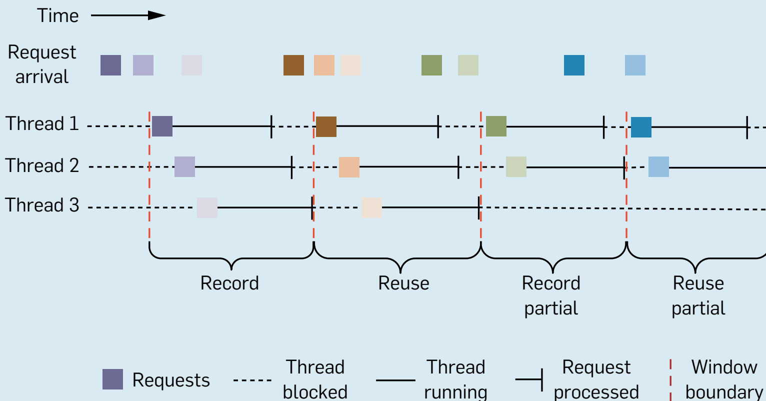


图 6.记录并重用三线程服务器程序的调度方案。连续执行流分成多个请求窗口，Peregrine 跨窗口记录和重用调度方案。



（或者说客户端请求），每个请求在随机的时刻到达，造成不同的调度方案；第三，由于请求并非同时到达，因此 PEREGRINE 无法对照调度方案的先决条件预先检查它们。

服务器程序往往返回同样的非活跃状态，因此 PEREGRINE 可使用这些状态将连续的请求流拆分成为多个请求窗口（请见图 6）。具体来说，PEREGRINE 在请求到达时缓冲请求，直至收集到足够的请求来使所有工作线程繁忙；然后它运行工作线程来处理请求，同时缓冲新到达的请求，以避免窗口间的干扰。如果在预定义的超时前，PEREGRINE 无法收集到足够的请求，那么它将以不完整的窗口继续，以减少响应时间。通过将请求流分成多个窗口，PEREGRINE 可以跨窗口记录并重用调度方案，从而使服务器程序稳定。服务器的非活跃状态可能逐渐变化；例如，Web 服务器可能在内存中缓存请求。PEREGRINE 让开发人员标注查询缓存的函数，将返回值作为输入，并选择恰当的调度方案。将请求分为窗口会减低并发性，但根据我们的实验，开销中等。

稳定多线程实现更好的程序分析

可在很多方面应用 StableMT 来提高可靠性。这里说一下我们基于 PEREGRINE 构建的一个程序分析框架，此框架用来分析多线程程序，在程序分析领域，多线程程序的分析一直是未解的难题。

这个难题的核心是精度和覆盖率的折衷。在两种常见类型的程序分析中，静态分析（分析源代码，但不运行源代码）覆盖所有调度方案，但是不精确（例如，发出很多误报）。其原因是，静态分析必须过度近似巨大的调度方案数，因此可能分析更多的调度方案，包括那些在运行时不可能存在的调度方案。静态分析可能检测到很多“故障”，而这些故障属于不可能存在的调度方案，这一点不足为奇。动态分析（运行代码并分析执行过程）可精确找出程序故障，因为它看得到代码的精确运行时效果。但是，由于调度方案过多，因此其覆盖率很低。

幸运的是，StableMT 减少了可能存在的调度方案，使得新的、兼具静态分析和动态分析有事的程序分析方法成为可能（请见图 7）。它在编译时对很少量的调度方案静态分析并行程序，然后在运行时动态实施这些调度方案。因为只关注很少的调度方案，因此 StableMT 大大提高了静态分析的精度，并减少了误报；通过动态实施分析过的调度方案，StableMT 可保证高覆盖率。动态分析同样受益，因为 StableMT 大大提高了其覆盖率（覆盖率定义为检查过的调度方案与所有调度方案之比）。

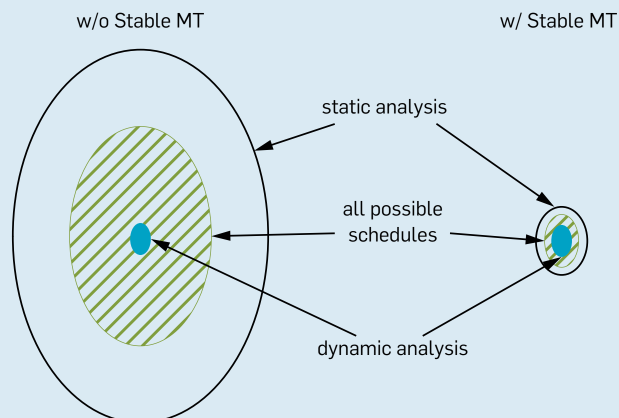
实现此方法的关键难题是，如何对一个调度方案进行静态程序分析。静态工具通常调用很多分析才能算出最终结果。为提高精度而修改工具的朴素方法是，修改每一个涉及的分析，不过这需要大量人力，而且很容易出错。这种方法可能还很脆弱；即，如果关键的分析步骤不是面向调度方案的，那么很容易造成最终结果不准确。

于是我们创建了新的程序分析框架和算法来根据调度方案对程序进行专门化，或者说智能转换。得到的程序的控制流和数据流比原来的程序简单，而且可以用常见的分析方法（如常量折叠和死代码消除）来进行分析，并显著提高精度。此外，我们的框架提供精确的“定义-使用”分析，可计算值在程序中是如何定义和使用的。其结果比常规“定义-使用”分析的结果精确得多，因为它只报告在运行时实施给定调度方案时，可能发生的实际情况。这样的精度可作为很多强力工具的基础，包括数据竞争检测器。

这里我们将说明我们框架的基本概念，我们先回想一下图 2 的示例。假设工具开发人员希望构建一个静态的数据竞争检测器，该检测器会标出什么时候不同的线程会并发写入同一个共享内存位置。虽然不同的工作线程访问分离的文件块，但现有的静态分析可能无法确定这一事实；例如，由于 nthread（线程数）是在运行时确定的，因此静态分析通常必须将动态线程估计为一个或两个抽象线程实例。所以它可能将不同线程对不同块的访问折叠为同一个访问，从而产生数据竞争的误报。

图 7.有/无 StableMT 时的程序分析。

若无 StableMT，静态分析所分析的调度方案往往比所有可能的调度方案还要多得多；动态分析倾向于分析所有可能的调度方案中的一小部分。StableMT 减少了调度方案，从而同时改进了静态分析和动态分析。



StableMT 简化了这些问题。假设每当 nthread 为 2 时，StableMT 总是实施图 3 中的调度方案。由于线程数是固定的，因此我们的程序分析框架重写示例程序，将 nthread 替换为 2。然后它展开循环，并克隆函数 worker，目的是向每个工作线程给予该线程自己的 worker 副本，这样就可以自动区分不同的工作线程。

我们的框架还可用来构造很多高覆盖率、高精度的分析；例如，我们的静态争用检测器在由以前的工具全面检查过的程序中，发现了七个以前未知的有害争用。它生成的误报极少，18 个程序中有 10 个程序没有误报，与其他静态争用检测器相比，误报数大幅下降。

评估

这里我们将介绍 PEREGRINE 的主要结果，重点落在两个评估问题上：

*它能否频繁重用调度方案？*重用率越高，程序行为变得越稳定，而 PEREGRINE 也越高效；

*它能否高效实施调度方案？*低开销对于频繁重用调度方案的程序非常重要。

我们选择了一组 18 个各不相同的程序作为我们的评估基准，这些程序要么是广泛使用的现实并行程序（如 Apache 和 PBZip2），要么是标准基准套件中计算密集型算法的并行实现。

稳定性。为了评估 PEREGRINE 的稳定性，或者说 Peregrine 能够多频繁地重用调度方案，我们将其计算出来的先决条件与根据手动检测得出的可能最佳的先决条件进行比较。表 1 包含某些手动得出的先决条件；在这 18 个程序中，Peregrine 为其中 9 个程序算出的先决条件与最佳先决条件一样好或接近，从而允许频繁重用，而对于其他程序，算出的先决条件限制性更强。

我们还测量给定工作负载下的调度方案重用率来评估稳定性（请见表 2 获得从 TERN 获取的，并且可在 PEREGRINE 中复制的结果）。这四个工作负载是我们收集的真实工作负载，或者开发人员使用的人造工作负载。[10]对于其中的三个工作负载，TERN 只使用了少量调度方案就处理了超过 90% 的工作负载。对于 MySQL-tx，TERN 的重用率大大降低，因为工作负载过于随机，难以重用调度方案。但是，它仍处理了 44.2% 的工作负载。

效率。实施调度方案的开销对于频繁重用调度方案的程序至关重要。图 8 显示了 TERN 和 PEREGRINE 的这一开销；每根条形表示执行时间，TERN 和 PEREGRINE 规范化为传统多线程，时间是 500 多次运行的平均时间；图 8 报告了 Apache 的吞吐量 (TPUT) 和响应时间 (RESP)。

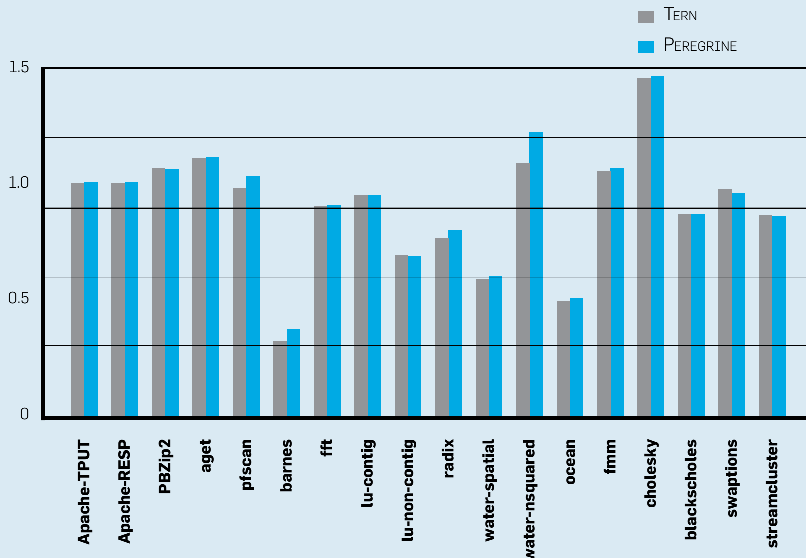
关于此图，可以得出两点结论：第一，大多数程序的开销不足 15%，说明 StableMT 可以非常高效。有两个程序（water-nsquared 和 cholesky）的开销相对较大，因为它们在紧密循环中执行大量互斥操作。即使这样，其开销仍小于 50%，远远低于以前 DMT 系统 1.1–10 倍的开销。[4] 某些程序得到加速，因为 TERN 和 PEREGRINE 安全地跳过了某些阻塞操作。[10,11]第二，PEREGRINE 只是比 TERN 稍慢，说明完全确定性也可以

表 2.四个工作负载下的调度方案重用率；“调度方案”列中列出了调度方案缓存中的调度方案数。

程序工作负载	重用率 (%)	调度方案
Apache-trace	90.3%	100
MySQL-simple	94.0%	50
MySQL-tx	44.2%	109
PBZip2-usr	96.2%	90

图 8.重用调度方案时的规范化执行时间。

值大于（或小于）1 的条形表示与传统多线程相比，速度减慢（或加快）。大多数程序重用调度方案的开销小于传统多线程总执行时间的 15%，而其他两个程序最多 50%。五个程序运行更快是因为 TERN 或 PEREGRINE 安全地跳过了某些阻塞操作。



很高效。（回想一下，Tern 只调度同步，而 Peregrine 还要调度内存访问，以使数据竞争变得确定。）

总结

通过构思、构建、应用和评估 StableMT 系统，我们证明了 StableMT 可以稳定程序行为，实现更高的可靠性，所以它工作起来高效且确定，同时大大提高静态分析的精确度。此外，它有望帮助解决并行编程简易化的重大难题。不过，Tern 和 Peregrine 仍只是研究原型，尚不适合普遍采用。而且，我们摸索出的想法只是 StableMT 在这个方向上开了个头；后面还有大量工作要做：

系统。在系统层，我们能不能构建高效、轻量级的 StableMT 系统，自动处理所有多线程程序？Tern 和 Peregrine 需要记录执行过程和分析源代码，这可能需要大量计算。随着内核数量的增加，研究者能不能构建可放大到数百个内核的 StableMT 系统？

应用。在应用层，我们只是触及了皮毛。改进程序分析只是一种可能的应用；其他应用包括提高测试覆盖率、仅对于少量在动态使用的调度方案来验证程序，以及基于调度方案优化线程调度和摆放，因为 StableMT 有效预测了未来。另外，稳定调度方案的想法可适用于其他并行编程方法（如 MPI、OpenMP 和类似 Cilk 的任务）。

概念。在概念层，我们能不能重做并行编程，以大大减少调度方案？例如，多线程系统可能默认不允许调度方案，只允许调度方案的开发人员显式编写代码来启用调度方案。因为开发人员具备一系列技能，我们可以只让最优秀的开发人员决定要使用什么调度方案，从而降低编程错误的可能性。

所以我们邀请所有人与我们一起，共同探索稳定多线程和可靠并行领域中这一前景广阔且令人兴奋的方向。

鸣谢

我们感谢为本文提供帮助的所有人。[10,11,22,25]尤其是，在 Peregrine 中构建别名分析的 John Gallagher、评估的 Chia-Che Tsai，以及帮助评估 Peregrine 的 Huayang Guo。我们还要感谢 Al Aho、Remzi Arpaci-Dusseau、Tom Bergan、Emery Berger、Luis Ceze、Xi Chen、Jason Flinn、Roxana Geambasu、Gail Kaiser、Jim Larus、Jinyang Li、Jiri Simsa、Ying Xu 以及很多匿名审阅者，感谢他们大量有帮助的评论。本文部分得到国家科学基金会的支持，包括 NFS Career Award、Air Force Research Laboratory、Air Force Office of Scientific Research Young Investigator Research Program Award、Office of Naval Research 以及 Sloan Research Fellowship。

参考资料

1. Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D., and Yelick, K. A view of the parallel computing landscape.*Commun. ACM52*,10(Oct.2009),56–67.
2. Aviram, A., Weng, S.-C., Hu, S., and Ford, B. Efficient system-enforced deterministic parallelism.*Commun. ACM55*,5(May2012),111–119.
3. Ball, T. and Rajamani, S.K.Automatically validating temporal safety properties of interfaces.In *Proceedings of the Eighth International SPIN Workshop on Model Checking of Software* (Toronto, May 19–20, 2001),103–122.
4. Bergan, T., Anderson, O., Devietti, J., Ceze, L., and Grossman, D. CoreDet:A compiler and runtime system for deterministic multithreaded execution.In *Proceedings of the 15th International Conference on Architecture Support for Programming Languages and Operating Systems* (Pittsburgh, PA, Mar. 13–17).ACM Press, New York, 2011,53–64.
5. Berger, E., Yang, T., Liu, T., Krishnan, D., and Novark, A. Grace:Safe and efficient concurrent programming. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications* (Orlando, FL, Oct. 25–29).ACM Press, New York, 2011,81–96.
6. Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., and Engler, D. A few billion lines of code later:Using static analysis to find bugs in the real world.*Commun. ACM53*,2(Feb.2010),66–75.
7. Bocchino, Jr., R.L., Adve, V.S., Dig, D., Adve, S.V., Heumann, S., Komuravelli, R., Overbey, J., Simmons, P., Sung, H., and Vakilian, M. A type and effect system for deterministic parallel Java.In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications* (Orlando, FL, Oct. 25–29). ACM Press, New York, 2011,97–116.
8. Cantrill, B. and Bonwick, J. Real-world concurrency. *Commun.ACM51*,11(Nov.2008),34–39.
9. Clarke, E., Grumberg, O., and Peled, D. *Model Checking.* MIT Press, Cambridge, MA, 1999.
10. Cui, H., Wu, J., Tsai, C.-C., and Yang, J. Stable deterministic multithreading through schedule memorization.In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation.* USENIX Association, Berkeley, CA, 2010.
11. Cui, H., Wu, J., Gallagher, J., Guo, H., and Yang, J. Efficient deterministic multithreading through schedule relaxation.In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles.*ACM Press, New York, 2011,337–351.
12. Devietti, J., Lucia, B., Ceze, L., and Oskin, M. DMP:Deterministic shared memory multiprocessing. In *Proceedings of the 14th International Conference on Architecture Support for Programming Languages and Operating Systems* (Washington, D.C., Mar. 7–11).ACM Press, New York, 2011,85–96.
13. Lee, E.A.The problem with threads.*Computer 39*, 5 (May 2006), 33–42.
14. Liu, T., Curtsinger, C., and Berger, E.D. DTHREADS:Efficient deterministic multithreading.In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles* (Cascais, Portugal, Oct. 23–26). ACM Press, New York, 2011,327–336.
15. Lu, S., Park, S., Seo, E., and Zhou, Y. Learning from mistakes:A comprehensive study on real-world concurrency bug characteristics.In *Proceedings of the 13th International Conference on Architecture Support for Programming Languages and Operating Systems* (Seattle, Mar. 1–5).ACM Press, New York, 2011,329–339.
16. Musuvathi, M., Qadeer, S., Ball, T., Basler, G., Nainar, P.A., and Neamtiu, I. Finding and reproducing Heisenbugs in concurrent programs.In *Proceedings of the Eighth Symposium on Operating Systems Design and Implementation* (San Diego, Dec. 8–10).USENIX Association, Berkeley, CA, 2008, 267–280.
17. Nethercote, N. and Seward, J. Valgrind:A framework for heavyweight dynamic binary instrumentation.In *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation* (San Diego, June 11–13).ACM Press, New York, 2011,89–100.
18. O' Hanlon, C. A conversation with John Hennessy and David Patterson.*Queue* 4, 10 (Dec. 2006), 14–22.
19. Olszewski, M., Ansel, J., and Amarasinghe, S. Kendo:Efficient deterministic multithreading in software.In *Proceedings of the 14th International Conference on Architecture Support for Programming Languages and Operating Systems* (Washington, D.C., Mar. 9–11).ACM Press, New York, 2011,97–108.
20. Shao, Z. Certified software.*Commun. ACM53*,12(Dec.2010),56–66.
21. Wu, J., Cui, H., and Yang, J. Bypassing races in live applications with execution filters.In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation* (Vancouver, Canada, Oct. 4–6). USENIX Association, Berkeley, CA, 2010.
22. Wu, J., Tang, Y., Hu, G., Cui, H., and Yang, J. Sound and precise analysis of parallel programs through schedule specialization.In *Proceedings of the ACM SIGPLAN 2012 Conference on Programming Language Design and Implementation* (Beijing, June 11–16).ACM Press, New York, 2011,205–216.
23. Xiong, W., Park, S., Zhang, J., Zhou, Y., and Ma, Z. Ad hoc synchronization considered harmful.In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation* (Vancouver, Canada, Oct. 4–6). USENIX Association, Berkeley, CA, 2010.
24. Yang, J., Cui, A., Stolfo, S., and Sethumadhavan, S. Concurrency attacks.In *Proceedings of the Fourth USENIX Workshop on Hot Topics in Parallelism* (Berkeley, CA, June 7–8).USENIX Association, Berkeley, CA, 2012, 15.
25. Yang, J., Cui, H., and Wu, J. Determinism is overrated:What really makes multithreaded programs hard to get right and what can be done about it?In *Proceedings of the Fifth USENIX Workshop on Hot Topics in Parallelism* (San Jose, CA, June 24–25). USENIX Association, Berkeley, CA, 2013.

Junfeng Yang (http://www.cs.columbia.edu/~junfeng) 是纽约哥伦比亚大学计算机科学系软件系统实验室的副教授兼副主任。

Heming Cui (http://www.cs.columbia.edu/~heming) 是纽约哥伦比亚大学计算机科学系软件系统实验室的博士生和成员。

Jingyue Wu (http://www.cs.columbia.edu/~jingyue) 是纽约哥伦比亚大学计算机科学系软件系统实验室的博士生和成员。

Gang Hu (http://www.cs.columbia.edu/~ganghu) 是纽约哥伦比亚大学计算机科学系软件系统实验室的博士生和成员。

Yang Tang (http://ytang.com) 是纽约哥伦比亚大学计算机科学系软件系统实验室的博士生和成员。

译文责任编委：陈文光

© 2014 ACM 0001-0782/14/03 $15.00

DOI:10.1145/2566590.2566610

与古代相比，如今秘密数据的嵌入方法更加复杂，不过基本原理依然没变。

ELZBIETA ZIELINSKA、WOJCIECH MAZURCZYK、KRZYSZTOF SZCZYPIORSKI

隐写术的趋势

由于危及私人公司和政府部门数据安全的事件屡屡曝光，大众媒体将 2011 年称为“黑客之年”[23]。实际上，失窃数据量总量估计完全可以达到 PB 级别（即 100 万 GB）。[2]

该年的大部分网络入侵都可以归咎于所谓的“暗鼠行动 (Operation Shady RAT)”。[49] 这些行动瞄准全世界许多机构，在很多情况下，所造成的破坏会持续数月。感染机制主要是诱使不知情的用户打开一封特别设计的电子邮件（网络钓鱼邮件），并在受害者的计算机上植入一个后门。接下来，受感染的计算机会链接到某个网站并下载一些看似合法的 HTML 文件或 JPEG 文件。网络罪犯们真正的行为是将指令编码到图片或精心制作的网页中，使这些命令对不知情的第三方隐藏，并且偷偷穿过防火墙进入受攻击的系统。然后，这些控制指令操纵受害人的计算机从远程服务器获取可执行代码，从而使外部人可以访问被入侵主机上的本地文件。32 在许多情况下，这些获取机密资源的通道会存在数月，因此人们认为这种安全漏洞非常严重。这些罪犯明目张胆，他们甚至没怎么努力掩盖攻击中使用了信息隐藏技术这一事实。著名的“Lena”是用作控制指令载体的图片之一，这张裁剪过的花花公子模特照片是所有数字图像处理或隐写算法的标准测试图像。

从而使外部人可以访问被入侵主机上的本地文件。[32] 在许多情况下，这些获取机密资源的通道会存在数月，因此人们认为这种安全漏洞非常严重。这些罪犯明目张胆，他们甚至没怎么努力掩盖攻击中使用了信息隐藏技术这一事实。著名的“Lena”是用作控制指令载体的图片之一，这张裁剪过的花花公子模特照片是所有数字图像处理或隐写算法的标准测试图像。

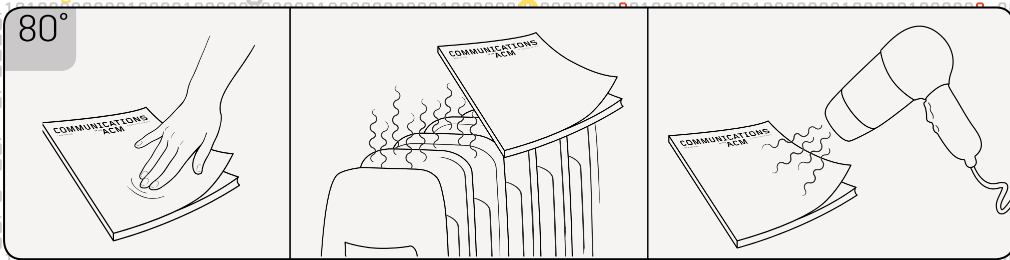
我们很有可能正在目睹一种全新恶意软件的诞生。这一切都始于 2010 年 6 月发现著名的震网 (Stuxnet)[18] 计算机蠕虫病毒，它由于专门针对伊朗核电站而备受关注。[14] 2011 年 9 月，人们发现了新的蠕虫 Duqu，它似乎与 Stuxnet 密切相关。[7] 上述两例中，恶意软件结构的一般特征相同，但与前者不同，Duqu 是为了收集受感染系统的信息。在 Duqu 错综复杂的功能中，最令人震惊的是它采用了一种特殊方式将获取的数据转移到恶意软件作者的指令与控制中心。获取的信息隐藏在看似无害的图片中并作为

» 重要见解

- **最近针对全球重要目标的攻击证明，隐写术，也即在合法载体内掩盖隐藏信息的存在，成为了恶意软件提供者的交易工具。**
- **尽管数百年前人们就知道隐写术了，但它最近扩散到了新的领域—数字媒体、计算机网络和热门通信服务。**
- **除了仔细搜索任何可能被用于嵌入非法信息的漏洞，或任何用逃避人类感知的方式改变潜在载体的方式之外，没有什么神奇的方法可以应对隐写术的滥用。**

美术设计：ANDRIJ BORYS ASSOCIATES

A P P L Y H E A T



普通文件在全球网络中传输，而不会引起任何怀疑。[21,51] 几乎同时被发现的恶意软件 Alureon 的一个新变种也使用了类似的功能机制 [3]。

这些事实表明，在当今数字科技世界，很容易想象得到，那些嵌入秘密数据的载体并不一定得是一张图片或是一段网页源代码，它也可能是计算机网络中自然出现的其它文件类型或数据组织单元，例如，一个数据包或一帧图像。然而，我们要强调，将秘密信息嵌入看似无害的载体这一过程并不是新发明，人类很久以前就已经知道并使用了这种方法。这一过程称为*隐写术 (steganography)*，其起源可以追溯到古代。而且，自出现之日起，它一直都在发挥着重要作用。

隐写术的用途之一是提供秘密通信方式。建立此类信息交换的目各不相同，用途可能是合法的活动，也可能是非法的活动。从明显的犯罪通信到防范系统的信息泄露，再到网络武器交易乃至工业间谍活动，人们经常强调隐写术非法的一面。另一面便是合法用途，包括规避网络审查和监控、[55] 计算机取证（追踪和识别）和版权保护（水印图像，广播监控）。

Stuxnet、Duqu、Alureon 和 Shady RAT 只是安全专家日常处理的一些例子。更应当让我们敲响警钟的是恶意黑客将隐写术纳入了自己的攻击武器库中。我们可以断定，隐写术将成为黑帽黑客 (Black Hat) 的新攻击手段。

隐写术的对立面 -- 隐写分析，最近才刚刚出现，它集中在检测隐秘通信。这一点可以从可用的关注信息隐藏的软件工具的比例表明。数据嵌入程序远远多于用于检测和提取嵌入内容的程序。最大的隐写工具商业数据库包含 1025 种应用程序（截至 2012 年 2 月）[4]，比 2007 年 Hayati 等人 [24] 提到的 111 种工具有所增加。

下面让我们仔细研究一下该技术的演化历程，特别是那些属于网络隐写术范畴的方法。

最近的隐写术应用案例

除了上文提到的隐写方法应用案例外，过去十年，我们可以发现大量关于隐写术及其检测方法（隐写分析）的研究工作。这有两方面的原因：首先，工业界和商业界对数字版权管理 (DRM) 的兴趣；其次，2011 年 9 月 11 日计划攻击美国的恐怖分子据称利用了隐写术。[29,45] 据称恐怖组织使用图像隐藏谋划指示，然后发布在公开网站上。[45] 这种通信方式似乎有长达三年未引起人们注意。[28]

最近的发现表明，隐写术目前主要用于非法用途。[29,44,50,57] Robin Bryant[11] 回顾了“Operation Twins”行动的案例，这一行动以 2002 年“Shadowz Brotherhood”犯罪团伙被捕而告终，该团伙是一个借助隐写术分发儿童色情物品的恋童癖组织。

涉及使用信息隐藏的事件迅速增多促使该问题获得官方的重视。2006 年的联邦报告 (Federal Report) 称，[38] 隐写术是当今网络的重大威胁之一，预计其重要性会继续上升。降低该技术相关风险的解决方案之一是熟悉隐写术的演化，从而预测其未来的发展趋势。20 世纪 80 年代初，隐写术刚刚开始普及，学术界就已经认识到这种需要。

事实证明，隐写方法是数据渗漏的实用工具，例如，据报道 [1]，2008 年美国司法部有人将数据嵌入多个图像文件中，把机密的财务数据从该机构偷了出去。

2010 年，号称“illegals”的俄罗斯间谍网曝光，表明隐写术可以在更长时间内不被察觉。该间谍组织利用数字图像隐写术从美国向莫斯科泄露机密信息。[44]

隐写术及其与密码学的关系

经常有人错误地将隐写术与信息隐藏互换使用。隐写术是一种将秘密信息嵌入（隐写）特定载体的技术，目的可能是进行隐秘通信。由于其定义难以捉摸，并且人们对发明的秘密通信方式，将其归属于隐写术或信息隐藏的特定领域，缺乏一致的分类，所以这两个领域之间的界限并不分明。产生误解的原因可能是大众媒体最近对隐写术的兴趣大增，而这些媒体只揭示了现有技术的冰山一角。过去十年中，有关间谍活动和恐怖活动的报道 [29,44,50,57] 主要推升了这些与互联网和数字图像隐写术相关的信息隐藏技术。

剩下的问题是：如何提出规则来区分哪些属于隐写术范围？办法是提出一些特定条件，要归为隐写术范围必须符合这些条件。这些条件表述如下：

- 通过此类隐秘方式传输的信息被嵌入看似无害的载体中，用该载体作为隐藏内容的伪装。
- 应用隐写术的目的是隐秘地传输信息。
- 通信的保密性主要由以下因素保证：应用到所使用载体的算法的伪装能力，处理后的数据与掩饰物的整个合法实体（没有嵌入）混合得有多好；这可理解为抵御检测尝试的能力，检测尝试依赖于对所捕获的流量进行统计分析或对可疑信息的感知分析。

秘密信息的最佳载体必须具有两个特征。首先，它应当很普遍，使用此类载体本身不应被认

为异常。其次，与隐写插入相关的载体修改对不知道隐写程序的第三方应该是“隐藏”的。因此，如果附加信息的嵌入引起了载体的退化，那么其严重程度应当不至于引起怀疑。

隐写术不仅限于要隐藏正在发送消息这一事实，而且如果没有被检测到，还要让发送者和接收者“不可见”。它也应该具备匿名性和隐私性，这在现代社会是可以理解的期望。显然，隐写术的匿名潜力对个人、社会和国家构成了一种新的威胁，虽然它在保护隐私方面可视为有益处。益处与威胁之间的权衡涉及很多复杂的伦理、法律和技术问题。本文只考虑后者。

由于提供保密性这一共同目的，人们经常将隐写术与密码学混淆。倘若知道这些词语的词源，它们之间的差别就变得很明显。隐写术源自希腊文中的“掩蔽书写”，而密码学代表“秘密书写”。前者描述一种创造隐藏信道的技术，后者表示持续公开的信息交换，未授权方无法理解其中的信息内容。总之，一种方式是建立秘密信道，另一种是对信息本身加密。无论哪种方式，两种技术的共同目的是防止信息泄露，这正是使我们能够区分二者的方式。表 1 总结了密码学和隐写术之间的差别，表 2 总结了隐写术和水印之间的关系。

尽管上述几种通信保护方法有共同的历史背景，但只有密码学一直维持着强势地位。古希腊和古罗马时代是隐写术的黄金年代，之后随着时间的推移，隐写术逐渐边缘化。凭直觉而言，任何依赖自身保密性来提供秘密通信的保护方法，都不应得到宣传。因此，很少发现有报道谈及当代对隐写术的利用，但这并不表明它是一门被忽视的科学学科。

隐写术的起源

隐写术的灵感与在动物界和植物界观察到的现象密切相关。进化论早已证明，对众多物种而言，模仿是很好的保护和生存技能。采用另一种生物体的特征来伪装自己的存在，这种能力称为拟态。它可使某些生物提高自己的存活率。在自然界中寻求灵感的古希腊人已经把这样的模仿能力作为技艺的一种衡量方式。古人自然而然地选择了普通物体作为秘密信息的载体。实际的载体（甚至可能是活体）必须在从通信的一方传递到另一方的途中不会引起怀疑。

第一份关于使用隐写术的书面记载出自希腊历史学家希罗多德 (Herodotus)，这应该并不令人意外。报告中所述方法是将秘密消息伪装在野兔尸体中。[16] 这只动物原本要作为比赛奖品，并由一名冒充的猎人携带。通过这种方式可以将信息传递出去且不会引起不必要的怀疑。

这名历史学家的作品所引述的方法中，最著名的当属用木板通信。这些木板通常都涂有一层薄蜡，蜡上压印有文字。如果将文字永久性刻在木头上（隐藏信息的载体）然后涂上一层蜡，借助这样的媒介就能够秘密传递信息。此类物体会被作为未使用的木板传递，只有知情的接收者才会知道蜡层熔化后就会显现出文字。

希腊人的这些方法依赖于共同的模式，因而很容易实现：传递信息时利用了当时的人们认为非常普通的载体表面。随着人类文明和人们沟通方式的进步，新机会出现了。纸莎草的替代品羊皮纸普及开来，带来了一种新隐蔽物。羊皮纸的普及催生了互补隐写算法，这些算法可以利用新隐蔽物的特性。老普林尼 (Pliny the Elder) 被认为是隐显墨水的发明人，[47] 他提出使用 thithymallus 植物的汁液书写文字，这些文字

表1隐写术和密码学的特征比较。

		密码学	隐写术
目的		混淆通信内容	隐藏通信事实
特征	保密性	密文无法辨认	对不知情的观察者而言，嵌入的信息“不可见”
	通信安全性	依赖密钥的保密性	依赖嵌入方法的保密性
	鲁棒性保证	加密算法的复杂性	感觉隐形/统计隐形/符合协议规范
	攻击	容易检测/难提取	难检测/难提取
对策	技术	逆向工程	交换数据的持续监控和分析
	法律	密码技术出口法	严密的设备/协议规范

表2隐写术和水印的特征比较

		水印	隐写术
目的		保护载体	保护秘密信息
特征	保密性	不可见或感知可见性由需求决定	对不知情的旁观者而言，嵌入信息“不可见”
	鲁棒性类型	抵抗篡改或移除的鲁棒性	抵抗检测的鲁棒性
	信号处理的效果/随机误码/压缩	不得导致水印丢失	可能导致隐藏数据丢失
	载体类型	数字文件—音频、视频、文本或图像	任何采用数字技术表示数据的服务、协议、文件、环境

在干燥后会变得不可见。微妙的加热过程会导致墨水中包含的有机物炭化，然后变成褐色。

所有上述技术的共同点是要在载体中添加其他成分（附加功能），否则就无法以实体形式包含插入的元素。

古罗马发明的另一种隐写术是语义编码，也即不采用书写形式的秘密信息。古代史学家塔西佗 (Tacitus) 对距骨块 (Astragali) 很有兴趣，[48] 后者是一种用骨头制成的小方块。这些物体可以串成一串，可为其中孔的位置赋予含义。适当制作的物体可以作为玩具传递而不引起别人注意。

中世纪带来了信息隐藏技术的重大进步。中国人发明的纸在中世纪传到了欧洲，使人们有必要区分不同厂商的产品。纸张水印就是这样出现的。[43] 如今，数字水印和数字图像隐写术都是基于同样的原理。需要强调的是，文件水印现在被视为信息隐藏技术的一个独立分支。Petitcolas、Anderson 和 Kuhn[40] 在 1999 年的综述论文中衍生出了整个版权标记领域，水印是其中一个子类别。此类嵌入式水印缺少明显的通信属性，向通信参与者提供“隐匿性”的作用居次，而鲁棒性更为重要，因此，目前的概念不将其归类为隐写术。

纸的普及有着更深远的影响。隐写载体不再必须是实际物体，而是能够采取书写形式，载体文字本身会隐藏保密信息。中世纪的这些发明中得到普及是文本隐写方法，特别是藏头诗。该术语指的是首个字母或音节能够拼出一条消息的作品。这种文字隐写最著名的例子由多明我会神父弗朗切斯科·科隆纳 (Francesco Colonna) 创造，他于 1499 年将一段爱情告白藏在自己的书《寻爱绮梦》(Hypnerotomachia Poliphili) 中，这段爱情告白能从后续章节的第一个字母拼写出来。[15]

秘密信息的最佳载体必须具有两个特征。首先，载体应很普遍。其次，与隐蔽性插入相关的载体修改对不知道隐写程序的第三方应该是“隐藏”的。

中世纪的人发现，更绝妙的载体是语言本身。在这种方式中，嵌入过程出现在语法和语义中。语言隐写术可以是从上述文字隐写技术衍生而来，因为它依赖于书面（甚至可能是口头）语言，目的是欺骗不知情者的知觉。按照 Richard Bergmair[10] 的主张，语言隐写术的范围涵盖任何故意模仿特定语言特有字词的典型结构的技术。这可能涉及故意篡改自然语言的语法、句法和语义。任何包含上述修改的行为都应能够维持隐蔽文字无害的表象。

文艺复兴时代，意大利科学家吉安巴蒂斯塔·德拉·波尔塔 (Giambattista della Porta) 有了新的发明。他在 16 世纪详细阐述了如何将信息隐藏在煮熟的鸡蛋中：用明矾和醋的混合物制成的墨水在蛋壳上写字。这种溶液可以穿透蛋壳且不在表面留下任何痕迹，但蛋白上会出现变色点，将信息留在其表面，只有剥去蛋壳后才能读到。

启蒙时代，德国传教士 Gaspar Schott 追随了文艺复兴时期前辈们的足迹。在 1680 年出版的作品《*Schola Steganographica*》中，他阐述了如何利用乐谱作为隐藏数据载体。每个音符对应一个字母，只要没有人尝试演奏这些听起来稀奇古怪的旋律，它们看起来是无害的。

启蒙时代之后的工业革命带来了新的通信方式。报纸成为了最新信息流行而可靠的来源。在某些时候，报纸显然能充当完美的隐写载体。自从日报可以免费派送之后，通过在选定的字母上扎孔来编制秘

密消息就变得很方便。“报码”就是这样产生的。

人们对隐写术的兴趣日益增加，最初表现可以追溯到第一次世界大战、第二次世界大战和之后的冷战时期。这些事件中产生了微点拷贝（图像或文字的显微底片上插入标点符号）之类的隐写技术。[56]

两次世界大战期间是隐藏通信方案真正的繁荣期。第一次世界大战见证了各种隐形墨水的盛大回归。[27] 第二次世界大战标志性的事物是 Hedy Lamarr 和 George Antheil 的扩频通信专利。[34] 他们设计出了一种采用特殊多频集信号且抗干扰的鱼雷制导方法。控制信息分散在提供掩蔽的宽频带宽内。之后，在不同频率内嵌入信息这一创意在数字图像和音频隐写术领域得到了运用。

20 世纪的技术发展加快了更多复杂技术的发展。在这些发明中，有一种叫做“阈下信道”，它是基于隐写嵌入的加密密码。其主要原理是将内容插入数字签名中。尽管美国政府禁止出版隐写资料，古斯塔夫·西蒙斯 (Gustavus Simmons) 1984 年引入了这一概念。西蒙斯提出为在两个参与者之间受监控的公开通信添加一个隐写信道。该信道基于许多消息验证专用比特。这些比特充当隐写信道的容量，代价是减弱数字签名的消息验证功能。[46] 这样建立的隐写信道是可见的，但无法检测到。阈下信道利用加密协议作为隐写载体。

当代发展趋势

当代隐写技术利用了 20 世纪的发明—计算机和网络。所谓的数字隐写术呈现四大主要发展趋势：数字媒体隐写术、语言隐写术、文件系统隐写术和网络隐写术。

这里将解释和描述数字隐写术的这四个主要分支。必须强调的是，目前该领域的大部分研究都是针对数字媒体隐写术和网络隐写术。前者是取得了显著成就的成熟研究区域，因此当前该领域的探索不如最近出现的各种网络隐写术技术领域活跃。

数字媒体隐写术可追溯到 20 世纪 70 年代，当时研究人员专注于研究在数字图片中秘密嵌入签名的方法。研究人员提出了许多不同的方法，包括拼凑、最低有效位修改和纹理块映射编码方法。[8] 这些技术用于两类图像：经过有损压缩或无损压缩的图像，例如 JPEG 或 BMP 这两种最常见的图像格式。在数字图像中嵌入信息的算法可以根据所引起的改变的类型分类。据 Johnson 和 Jajodia 所述，修改可以逐位进行（影响图像的空间域特征）或影响频域特征。再次，可以利用特定文件格式的复杂性。当然，也可以综合所有这些技术。变换域提供了用途最广泛的嵌入媒介。影响图像处理的算法包括离散余弦变换 (DCT)、离散小波变换 (DWT)、傅里叶 (Fourier) 变换，这些算法可能会导致亮度或图像的其他可测量特征发生变化。[20,26] 数字图像隐写术的地位稳固。Cheddad 等人[13] 的调查论文指出，当前的兴趣集中于采用数字媒体隐写术和数字水印在医学影像中嵌入与患者相关的机密信息。数字图像隐写术的另一种用途预计会普及，那就是在印刷品中植入附加数据，这些数据肉眼不可见，用手机拍摄和处理后可以解码。[13]

值得注意的是，数字图像隐写术主要是欺骗人的视觉系统，让人以为所感知的图像未被以任何方式篡改。[8] 类似规则适用于整个数字媒体隐写术领域，其首要功能是欺骗观察者，让他们相信所制作的“赝品”确实是正品。整个隐写算法的通信属性次于秘密数据的嵌入过程。

随着数字图像隐写术的发展，人的听觉系统似乎和视觉感知一样容易受到欺骗。研究重心转向了 MPEG 等格式的音频文件。开发的技术包括频率掩蔽、回声隐藏、相位编码、混杂和扩频。显然，纠错编码是良好的音频隐写术补充载体。任何冗余数据都能传输隐写信息，代价是损失一些随机误差的鲁棒性。[8] 该创意后来用在了基于隐写术的网络协议中。

接下来，隐写术研究者采用了视频文件作为目标载体。他们提出的大部分方法是修改为音频和图片文件而提出的算法。针对视频的解决方案包括使用视频 I 帧的颜色空间[54] 或 P 帧和 B 帧的运动矢量作为隐写载体。[58] 目前，视频文件隐写术要么利用处理音频和图像文件的现有方法，要么利用视频传输的固有属性，例如运动编码。

与数字图像隐写术和音频隐写术一起，人们还发明了文本信息隐藏术，现有方法利用了文字的各个方面。第一代技术是改变字间距。据称，在撒切尔夫人 (Margaret Thatcher) 时代，该技术就已用于追踪秘密文件的泄露。[5] 更多先进的隐写方法使用了文字的句法和语义结构作为载体。上述方法可以对标点符号的移位、语序或同义词选择的变化赋予特定含义。现在有人提出，甚至垃圾邮件信息也可作为隐写术的载体，因为每天都有人发送大量此类邮件。[12] 根据 Bennet 的研究，[9] 这些可能的技术可以依靠生成带有紧密语言结构的文字或用自然语言作为载体。我们应当注

意到，第一批技术并不完全符合隐写术的定义，即载体应该独立于所注入隐藏内容而存在。因此，缺乏文本的修辞结构不能被视为合适的载体。

专家们还区分了文字隐写术和语言隐写术。[9]“垃圾邮件方法”是一种语言学方法，嵌入在上下文无关文法 (CFG) 的帮助下发生。CFG 为树形结构，因此选择适当的词或分支可以对二进制数据进行编码。文本方法的一个例子是替代技术，其中消息载体是经历了移位、重复或其他修改的空白和标点符号组合。

与此同时，有人披露，x86 机器代码也可以进行嵌入。[17] 在精心挑选的具有同等功能的指令帮助下，载体代码中可以放置一些信息。该方法与语言隐写术利用了相同的原理，可以对从一组同义词中选择词语赋予隐写含义。

Anderson、Needham 和 Shamir 发明的隐写文件系统让人大开眼界。[6] 显然，即便在隔离的计算环境下也能够隐写嵌入信息。隐写制作的主要原理类似于隐形墨水：知道如何进行搜索的人能够泄露磁盘上的加密文件。加密数据类似于磁盘上自然存在的随机位，并且只有能提取标记文件界限的矢量才允许定位，利用的机制依赖于这一点。Pang 等人 [39] 的文章介绍了隐写文件系统的另一个例子，作者创造了一个在 Linux 上实现的隐写文件系统。他们的发明保持了被存储文件的完整性，并且采用了磁盘空间隐藏方案，以虚拟隐藏文件和遗弃块作为伪装。

除了上文提到的数字隐写术类型外，目前更多人感兴趣的是网络隐写术。这一现代方法系列源自“秘密信道”，其中许多技术原本用于单片机系统，如大型机。此术语最初由 Lampson 提出，他发现了非受限程序中的信息泄漏问题。[31]“网络隐写术”这一表达由 Szczypiorski[52] 创造。目前，术语网络隐写术和秘密通道被互换使用（错误地），但历史上它们是互相独立的。

附图概括了隐写数据载体的演变。

网络隐写术：备受瞩目的最新技术

网络隐写术是信息隐藏术的最新分支。这是一个飞速发展的领域：近年来产生了多种新的信息隐藏方法，它们可以用于各种类型的网络。网络隐写术的本质是利用开放系统互连 (OSI) 参考模型 [59] 中的协议。这一系列方法可能会同时利用一个或多个协议或协议之间的关系，依赖于修改隐写嵌入的固有属性。

网络隐写术处于上升期，因为人们发现把秘密数据嵌入数字媒体文件有两个严重缺点：每个文件中只能隐藏有限数量的数据，并且修改后的图片可能会被取证专家获取（例如，因为它被上传到某种服务器上）。网络级嵌入完全改变了这一状况；它可以在很长时间内泄露信息（即使很慢），并且如果未截获所有交换的流量，就不会留下任何东西供取证专家分析。因此，此类方法更难检测和从网络中消除。

现在，网络隐写术依赖某些漏洞来掩饰其存在。首先，终端用户要无法分辨看似相同的物体之间的微小差别。例如，听通过公共网络传输的实时音频录音时，人们几乎肯定不会注意到所传输语音的轻微改变，特别是在缺少该特定 VoIP 电话的任何参考信息的情况下。其次，漏洞允许隐写消息通过网络，而不会在中间节点引发任何警报。这通常与统计隐形相关，也就是说，引起的异常不超过网络功能通常的合理阈值。一般而言，隐写利用了通信的三个特征：

- 信道并不完美。错误是一种自然现象，因此有可能模仿损坏的协议数据单元 (PDU) 的普通分发来嵌入信息。
- 大多数协议容忍一定量的冗余信息。只要不引发载波信息流故障，就可以用剩余字段来嵌入信息。
- 并非所有协议都完全固定。大部分规范允许实现时有一定自由，这点可以被滥用。

按 Jankowski 等人 [25] 所述，网络隐写方法可以按照隐写所使用的协议数量进行大致分类。修改 OSI 模型单个协议的属性称为协议内隐写，而利用多种协议之间的关系归为跨协议隐写。一旦选择了一种或多种协议作为秘密数据的载体，嵌入方式就已经确定。第一种可能性是将隐秘信息注入协议数据单元 (PDU)。这可以通过修改特定协议字段或插入有效载荷，或同时使用二者来完成。作为先前技术的替代或补充，可以修改 PDU 之间的时间关系。这些改变会影响 PDU 的顺序、丢失或相对延迟。混合方法同时利用这两种方式：修改 PDU 及其时间关系。

在当前更先进的网络信息隐写方法之前，人们利用 TCP/IP 栈协议 [42] 的不同字段作为隐藏数据载体。早期方法大多集中于在协议未使用或保留的字段中进行嵌入来传递秘密数据。之后，人们发明了更先进的方法，这些方法针对特定的环境或服务。最近的解决方案利用了：

- 多媒体、IP 电话等实时服务；[33]
- 热门点对点服务：Skype[35] 或 P2P 文件共享系统，例如 BitTorrent；[30]

- 社交媒体网站，例如 Facebook；[7]
- 无线网络环境：例如无线局域网 (WLAN)[53] 或长期演进技术 (LTE)；[22]
- 云计算环境[41]；和
- 新的网络协议，例如流控制传输协议 (SCTP)。[19]

虽然使用 IP 电话服务作为隐藏数据载体可视为最近的发现，[33] 现有 VoIP 隐写方法出自两种截然不同的研究起源。第一种是上文提到的数字媒体隐写术，它已经发展得比较成熟，产生了多种将所传输语音的数字表示作为隐藏数据载体的方法。第二种解决方案针对特定的 VoIP 协议（例如，信令协议、传输协议或控制协议）领域或协议行为。

最近，有人提出了转码隐写术 (TranSteg)，它适用于 IP 电话等多种多媒体和实时应用。[36]TranSteg 的基本思路是将音频数据从较高比特率的编解码器转码（有损压缩）到较低比特率的编解码器，从而减小语音有效载荷。这个过程通过压缩公开数据在有效载荷字段中为隐写信息腾出空间，语音质量只有最低限度的下降。获得的隐写带宽高达 32kbp/s。

在研究 P2P 服务的隐写适用性时，人们可能会遇到一种名为 SkyDe(Skype 隐藏) 的隐写方法，它由 Mazurczyk 等人针对 Skype 提出，[35] 利用加密的 Skype 语音数据包作为隐藏数据载体。通过利用话音激活率和数据包大小之间的高度相关性，可以识别未携带语音信号的数据包并用其携带秘密数据。实现方法是用秘密数据位替换加密的静音数据。产生的隐写带宽或隐藏数据速率（单位时间内使用特定方法可以发送的秘密数据量）约为 2kbps。

最近另一项针对互联网 P2P 服务的发明 StegTorrent 被引入了 BitTorrent 应用程序中。[30]BitTorrent 中通常为多对一传输，在其特有的 μTP 协议中，报头提供了一种为数据包编号并检索其原始序列的方法，StegTorrent 利用了这两点。这种方法能够以大约 270bps 的速率发送隐藏数据。

对 Facebook 之类的社交媒体网站，Nagaraja 等人[37] 提出创建在不可观测的信道上通信的僵尸网络。僵尸机器在图像中嵌入信息并使用图像分享功能将秘密数据路由给接收方，从而达到与僵尸主控机交换信息的目的。

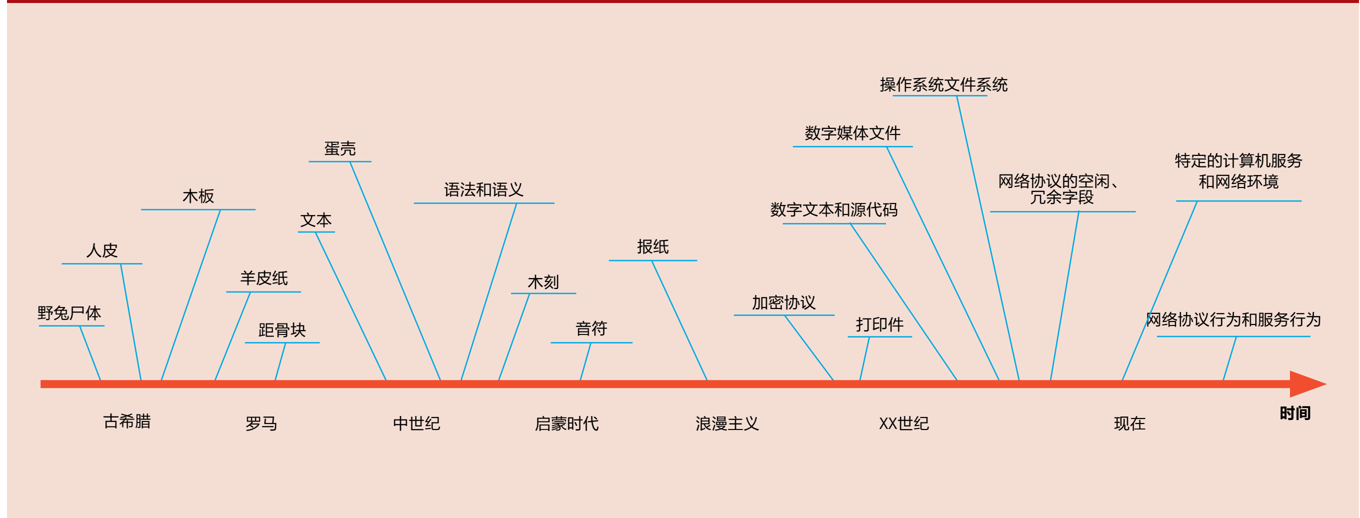
对于“无线环境”，隐写研究者们瞄向了不同的标准。例如，对于 WLAN，Szczypiorski 和 Mazurczyk 提出了一种称为无线填充 (WiPad) 的方法。[53] 该技术的原理是在 WLAN 物理层将隐藏数据插入帧的填充位中。这种方式能以高达近 1.5 Mbps 的速率秘密传输数据。Grabska 和 Szczypiorski[22] 对 LTE 利用了类似的概念，获得的的数据速率约为 1.2Mbps。

Ristenpart 等人[41] 视为易受跨虚拟机信息泄露攻击的云计算环境是施展隐写术的绝佳地方。他们提出了一系列技术，通过探测共享缓存负载、CPU 负载、击键活动或类似的方法来获取机密信息。

其他一些有前景的未来网络协议，例如 SCTP，也容易用于隐写术。SCTP 是新传输层协议的候选对象，可能会取代 TCP（传输控制协议）和 UDP（用户数据报协议）。Fraczek 等人[19] 的详细分析揭示了 SCTP 传输中最可能用于隐藏信息的地方。利用 SCTP 特有新功能（例如多宿主和多数据流）的隐写方法受到了特别关注。

总之，各种网络服务和应用程序都能够并且将成为嵌入目标。服务或应用程序普及得越广泛，就越吸引人通过网络隐写术附加秘密信息。

隐藏数据载体演化年表



结论

信息隐藏学的范围涵盖用于传达消息并使这种信息交换的某些方面保密的各种技术。这可能包括通过模糊对话参与者（匿名）、消息的保密性（隐写术）或载体保护（版权标记）保证安全性。

这些方法的根源可以追溯到有历史记载的时期。由于需要使发送的信息在被截获时无法破译，这促使人们创造看似无害但实际含义与表面上不同的代码或符号。

现代信息隐藏学采用了各种嵌入技术，其中许多是将先前已知的方法转移到数字领域的结果。有一个有趣的例外，那就是网络隐写术，它是随着网络环境的普及而出现的一系列方法。在隐写术中，新秘密数据载体的出现可视为信息隐藏技术发展的进化步骤。通信协议、服务和计算环境日益增加，这为展现整个隐写方法的范围创造了无限机会。值得注意的是，正是其属性以及普及程度注定或限制了载体作为有效秘密通信媒介的能力，新技术的出现很可能会创造新的信息嵌入方式。

最近的网络战事件显示，虚拟世界的非法活动对社会构成了实实在在的威胁。毋庸置疑，信息隐藏学这一武器已经得到利用，因此，人们应该意识到它对信息系统的安全性构成了重大威胁。更重要的是，形势很紧迫，因为隐写分析技术仍然落后最新的隐写方法一步。没有什么解决方案能“包治百病”并能检测我们目前网络安全防御系统中的秘密通信。因此，我们敦促研究界集中精力寻找可在网络环境中迅速实际部署的隐写分析方法。 C

现在，网络隐写术依赖某些漏洞来掩饰其存在。

参考资料

1. Adee, S. Spy vs. spy.*IEEE Spectrum.* (Aug. 2008); http://spectrum.ieee.org/computing/-software/spy-vs-spy/1.
2. Alperovitch, D. *Revealed:Operation Shady RAT.*McAfee, 2011; http://www.mcafee.com/-us/resources/white-papers/wp-operation-shady-rat.pdf.
3. Alureon trojan uses steganography to receive commands.(Sept. 2011); http://-www.virusbtn.com/news/2011/09_26.
4. Analysis, S. and Center, R. World’s largest digital steganography database expands again.*SARC Press Release* (Feb. 2012); http://www.sarc-wv.com/news/press_releases/-2012/safdb_v312.aspx.
5. Anderson, R. Stretching the limits of steganography.*Information Hiding.*Springer, 1996, 39–48.
6. Anderson, R. Needham, R. and Shamir, A. The steganographic file system.*Information Hiding.*Springer, 1998, 73–82.
7. Bencsáth, B., Pék, G., Buttyán, L. and Félegyházi, M. Duqu:A Stuxnet-like malware found in the wild, (2011); http://www.crysys.hu/publications/files/-bencsathPBF11duqu.pdf.
8. Bender, W., Gruhl, D., Morimoto, N. and Lu, A. Techniques for data hiding.*IBM Systems Journal 35*, 3&4 (1996), 313–336.
9. Bennett, K. Linguistic steganography:Survey, analysis, and robustness concerns for hiding information in text (2004).
10. Bergmair, R. A comprehensive bibliography of linguistic steganography.In *Proceedings of the SPIE Intl Conf. on Security, Steganography, and Watermarking of Multimedia Contents*, 2007.
11. Bryant, R., Ed. *Investigating Digital Crime.* John Wiley & Sons, 2008, 1–24.
12. Castiglione, A. De Santis, A., Fiore, U. and Palmieri, F. An asynchronous covert channel using SPAM.*Computers & Mathematics with Applications*, 2011.
13. Cheddad, A., Condell, J., Curran, K. and Mc Kevitt, P. Digital image steganography:Survey and analysis of current methods.*Signal Processing 90*, 3 (2010), 727–752.
14. Chen, T. Stuxnet, the real start of cyber warfare?*IEEE Network 24*, 6 (2010), 2–3.
15. Cox, I. *Digital Watermarking and Steganography.* Morgan Kaufmann, 2008.
16. De Sélincourt, A. *Herodotus:The Histories.* Penguin, 1954.
17. El-Khalil, R. and Keromytis, A. Hydan:Hiding information in program binaries.*Information and Communications Security*, (2004), 287–291.
18. Falliere, N., Murchu, L. and Chien, E. W32. Stuxnet dossier.White paper.Symantec Corp., Security Response, 2011.
19. Fra¸czek, W., Mazurczyk, W. and Szczypiorski, K. Hiding information in Stream

Control Transmission Protocol.*Computer Communications 35*, 2 (2012), 159–169.
20. Fridrich, J. *Steganography in Digital Media—Principles, Algorithms, and Applications.* Cambridge University Press, 2010.
21. Goodin, D. Duqu spawned by ' well-funded team of competent coders.' World ' s first known modular rootkit does steganography, too.*The Register*, Nov. 2011.
22. Grabska, I. and Szczypiorski, K. Steganography in LTE.In *Proc. of Intl.Workshop on Cyber Crime*, (San Jose, May 2014).
23. Gross, M.J.Exclusive:Operation shady RAT—Unprecedented cyber-espionage campaign and intellectual-property bonanza.*Vanity Fair* (Aug. 2011).
24. Hayati, P., Potdar, V. and Chang, E. A survey of steganographic and steganalytic tools for the digital forensic investigator.In *Workshop of Information Hiding and Digital Watermarking*, (Canada, 2007).
25. Jankowski, B., Mazurczyk, W., and Szczypiorski, K. PadSteg:Introducing inter-protocol steganography.*Telecommunication Systems:Modeling, Analysis, Design and Management 52*, 2 (2013), 1101–1111.
26. Johnson, N. and Jajodia, S. Steganalysis of images created using current steganography software.*Information Hiding.*Springer, 1998, 273–289.
27. Kahn, D. The history of steganography. *Information Hiding.*Springer, 1996, 1–5.
28. Kellen, T. Hiding in plain view:Could steganography be a terrorist tool?*SANS Institute InfoSec Reading Room*, 2001; http://www.sans.org/reading_room/whitepapers/-stenganography/hiding-plain-view-steganography-terrorist-tool_551.
29. Kelley, J. Terror groups hide behind Web encryption.*USA Today* (May 2001); http://-www.usatoday.com/tech/news/2001-02-05-binladen.htm.
30. Kopiczko, P., Mazurczyk, W. and Szczypiorski, K. StegTorrent:A steganographic method for P2P files sharing service.In *Proc. of Intl. Workshop on Cyber Crime*, (San Francisco, CA, May 2013).
31. Lampson, B. A note on the confinement problem.*Commun.ACM 16*, 10 (Oct. 1973), 613–615.
32. Lau, H. The truth behind the Shady RAT. *McAffe report*, (Aug. 2011); http://-www.symantec.com/connect/blogs/truth-behind-shady-rat.
33. Lubacz, J., Mazurczyk, W. and Szczypiorski, K. Vice over IP.*IEEE Spectrum 47*, 2 (2010), 42–47.
34. Markey, H. and Antheil, G. Secret communication system.Aug. 11 1942, US Patent 2,292,387.
35. Mazurczyk, W., Karas´, M. and Szczypiorski, K. SkyDe:A Skype-based steganographic method.*International J. Computers, Communications & Control 8*, 3 (June 2013), 389–400.
36. Mazurczyk, W., Szaga, P. and Szczypiorski, K. Using transcoding for hidden communication in IP telephony.*Multimedia Tools and Applications* (2011), 1–27; DOI 10.1007/s11042-012-1224-8.
37. Nagaraja, S., Houmansadr, A., Piyawongwisal, P., Singh, V. Agarwal, and Borisov, N. Stegobot:A covert social network botnet. *Information Hiding.* Springer, 2011, 299–313.
38. Networking and Information Technology Research and Development Program.I.W.G. on Cyber Security and I. Assurance.*Federal Plan for Cyber Security and Information Assurance Research and Development*, (Apr. 2006); http://www.nitrd.gov/pubs/csia/-csia_federal_plan.pdf.
39. Pang, H., Tan, K. and Zhou, X. Stegfs:A steganographic file system.In *Proceedings of the 19th Intl.Conf. on Data Engineering.* IEEE, 2003, 657–667.
40. Petitcolas, F., Anderson, R. and Kuhn, M. Information hiding—A survey.In *Proceedings of the IEEE 87*, 7 (1999), 1062–1078.
41. Ristenpart, T., Tromer, E., Shacham, H. and Savage, S.'Hey, you, get off of my cloud:Exploring information leakage in third-party compute clouds.'In *Proceedings of the 16th ACM Conference on Computer and Communications Security.*ACM, 2009, 199–212.
42. Rowland, C. Covert channels in the TCP/IP protocol suite.*First Monday 2*, 5 (1997).
43. Rudin, B. and Tanner, R. *Making Paper:A Look into the History of an Ancient Craft.* Rudin, 1990.
44. Shachtman, N. FBI:Spies hid secret messages on public websites.*Wired* (June 2010); http://www.wired.com/dangerroom/2010/06/alleged-spies-hid-secret-messages-on-public-websites/.
45. Sieberg, D. Bin Laden exploits technology to suit his needs.*CNN* (Sept. 2001); http://-edition.cnn.com/2001/US/09/20/inv.terrorist.search/.
46. Simmons, G. The prisoners' problem and the subliminal channel.In *Proceedings of Crypto '83:Advances in Cryptology* (1984), 51–67.
47. Singh, S. *The Code Book:The Secret History of Codes and Codebreaking.* Fourth Estate, 2000.
48. Smith, D. Number games and number rhymes:The great number game of dice. *The Teachers College Record 13*, 5 (1912), 39–53.
49. Srivastava, K. Congress wants answers on world ' s largest security breach.Aug. 2011; http://www.mobiledia.com/news/102480.html.
50. Stier, C. *Russian spy ring hid secret messages on the Web.*(July 2010); http://-www.newscientist.com/article/dn19126-russian-spy-ring-hid-secret-messages-on-the-web.html.
51. Symantec.W32.Duqu—The precursor to the next Stuxnet, (Nov. 2011); http://-www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/-w32_duqu_the_precursor_to_the_next_stuxnet_research.pdf.
52. Szczypiorski, K. Steganography in TCP/IP networks.In *Proceedings of State of the Art and a Proposal of a New System–HICCUPS.* Institute of Telecommunications' seminar, Warsaw University of Technology, Poland, 2003.
53. Szczypiorski, K. and Mazurczyk, W. Steganography in IEEE 802.11 OFDM symbols.*Security and Communication Networks 3* (2011), 1–12.
54. Wang, Y. and Izquierdo, E. High-capacity data hiding in MPEG-2 compressed video. In *Proceeding of the 9th Intl.Workshop on Systems, Signals and Image Processing* (Manchester, U.K., 2002), 212–218.
55. Wayner, P. *Disappearing Cryptography—Information Hiding:Steganography & Watermarking.*Morgan Kaufmann, 2009.
56. White, W. *The Microdot:History and Application.* Phillips Publications, 1992.
57. Williams, C. Russian spy ring bust uncovers tech toolkit.*The Register* (June 2010); http://www.theregister.co.uk/2010/06/29/spy_ring_tech/.
58. Xu, C., Ping, X. and Zhang, T. Steganography in compressed video stream.In *Proceedings of the First International Conference on Innovative Computing, Information and Control.*IEEE, 2006, 269–272.
59. Zimmermann, H. OSI reference model—the ISO model of architecture for open systems interconnection.*IEEE Transactions on Communications 28*, 4 (1980), 425–432.

Elz˙bieta Zielin´ska (ezielinska@tele.pw.edu.pl) 是波兰华沙理工大学电信研究所的研究助理。

Wojciech Mazurczyk(wmazurczyk@cygnus.tele.pw.edu.pl) 是波兰华沙理工大学电信研究所的助理教授。

Krzysztof Szczypiorski (ksz@tele.pw.edu.pl) 是波兰华沙理工大学电信研究所的教授。

译文责任编辑：孙晓明

© 2014 ACM 0001-0782/14/03 $15.00

第98页
技术视角
智能手机安全“污点跟踪”的前世今生
作者：Dan Wallach

第99页
TaintDroid：用于在智能手机上实时监控隐私数据的信息流跟踪系统
作者：William Enck、Peter Gilbert、Byung-Gon Chun、Landon P. Cox、Jaeyeon Jung、Patrick McDaniel 和 Anmol N. Sheth

DOI:10.1145/2576234

如需查阅随附论文，请访问 doi.acm.org/10.1145/2494522 rh

技术视角
智能手机安全“污点跟踪”的前世今生

作者：Dan Wallach

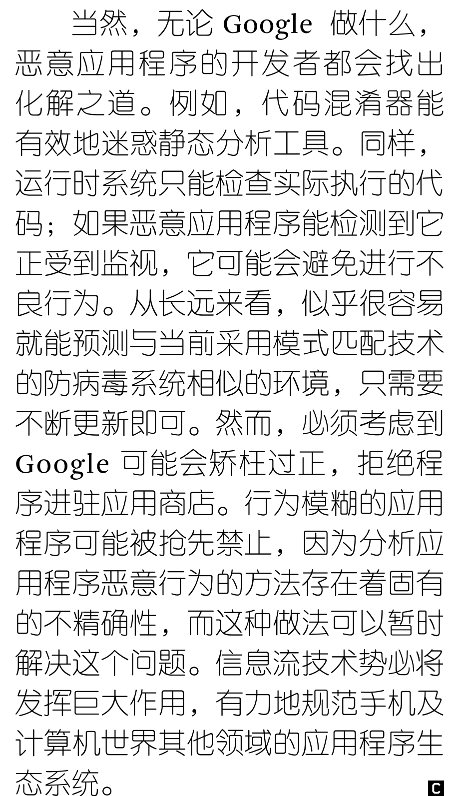
信息流作为一项安全策略，自有其诱人之处。你可以制定一份明确而简洁的策略，比如，“禁止我的 GPS 位置信息流入网络”，这似乎更符合我们的是非观。而 iOS 和 Android 等智能手机操作系统目前所尝试执行的策略则不然；它们就像在说：“把你的 GPS 位置信息告诉这款应用程序，你只能选择同意或不同意，而且你无权决定此类信息的使用方式”。信息流研究可以追溯至 20 世纪 70 年代早期。尽管许多最初的计算机科学理论和系统在研发时都模拟了军队关于处理秘密、机密和绝密数据的规定，但信息流策略和技术在当今时代具有重要意义，而且我们可以从这些早期研究中受益。

Dorothy 和 Peter Denning 在 1977 年的《*ACM 通讯*》中发表了题为“安全信息流的认证程序”的文章，并在文中提出静态分析策略。这是一篇具有里程碑意义的论文，也是一篇优秀的奠基性参考文献。[a]与此同时，其他人则寻求硬件解决方案或基于运行时的解决方案。就像现在一样，当程序的控制流取决于敏感数值时（比如，“如果我的 GPS 位置位于华盛顿特区，就采取不同的行为”），情况就会更加复杂，更别提还有异常控制流（例如，中断处理程序、异常，间接分支）和模糊的数据引用（指针解引用，数组索引）。在硬件中跟踪所有这些信息需要额外的计算和状态；而进行静态分析又会在执行路径上引起误报，而这在运行时中永远不会发生。

时至今日，出现了哪些新的局面？如今，我们拥有迅如闪电的计算机，就连手机也是如此，应用程序在绝大部分时间处于空闲状态，等待用户输入或网络数据。我们无需浪费过多电量或降低用户体验，就能承受额外的运行时 CPU 开销。同样，更好的算法和更快的电脑使得用于查找错误或安全验证的全程序静态分析发展为价值数十亿美元的产业。

TaintDroid 项目采用运行时污点跟踪方法来分析 Android 应用程序。这种方法的好处在于，用于分发应用程序的传输格式（Dalvik VM 的字节码）不是原始机器码，可以通过修改手机上的 Dalvik 编译器来添加运行时污点跟踪，让 CPU 密集型工作负载的性能开销处于 14%，这是一个完全合理的水平。与此类似，他们还将注释添加到 Android 文件系统和 IPC 层来跟踪受污染数据。TaintDroid 团队确实走了捷径：他们将并不跟踪原生 ARM 代码（一些 Android 应用程序可能包含此类代码以提高性能），他们同样也不跟踪从条件表达式到相关计算的污点。（后果：恶意应用程序经简单设计后，就可以欺骗当前的 TaintDroid 实现方案，要想强化 TaintDroid 来解决这一问题，就需要静态分析、更多的运行时开销，或二者皆有。）

尽管存在这些局限性，TaintDroid 项目仍成功地识别了许多行为方式会令用户明显反感的 Android 应用程序。（事后看来，你安装了某款由广告支持的免费应用程序，它们当然希望尽量获得关于你的更多信息，以便更有效地向你投放广告，这没什么好惊讶的。）自 TaintDroid 于 2010 年发布以来，业界不断致力于研究各种各样的方法来加强 Android 的安全性——无论是通过静态分析还是动态运行时机制。就连 Google 也加入了这场游戏，它在 2012 年发布了“Bouncer”系统，该系统在 Android 应用商店内运行，采用将静态分析与在虚拟机中运行每款应用程序相结合的方法来检测不良行为。Google 自己并未太多地透露 Bouncer 的工作原理[b]，但 Miller 和 Oberheide 却开展了一些巧妙的逆向工程，并揭示 Google 仍有很长的路要走。[c]



当然，无论 Google 做什么，恶意应用程序的开发者都会找出化解之道。例如，代码混淆器能有效地迷惑静态分析工具。同样，运行时系统只能检查实际执行的代码；如果恶意应用程序能检测到它正受到监视，它可能会避免进行不良行为。从长远来看，似乎很容易就能预测与当前采用模式匹配技术的防病毒系统相似的环境，只需要不断更新即可。然而，必须考虑到 Google 可能会矫枉过正，拒绝程序进驻应用商店。行为模糊的应用程序可能被抢先禁止，因为分析应用程序恶意行为的方法存在着固有的不精确性，而这种做法可以暂时解决这个问题。信息流技术势必将发挥巨大作用，有力地规范手机及计算机世界其他领域的应用程序生态系统。

a http://dl.acm.org/citation.cfm?id=359712

b http://googlemobile.blogspot.com/2012/02/android-and-security.html

c https://blog.duosecurity.com/2012/06/dissecting-androids-bouncer/

Dan Wallach (dwallach@cs.rice.edu) 是莱斯大学计算机科学系系统组的教授，该学校位于德克萨斯州休斯顿市。

译文责任编辑：陈海波

版权归作者所有。

DOI:10.1145/2494522

TaintDroid：用于在智能手机上实时监控隐私数据的信息流跟踪系统

作者：William Enck、Peter Gilbert、Byung-Gon Chun、Landon P. Cox、Jaeyeon Jung、Patrick McDaniel 和 Anmol N. Sheth

摘要

如今的智能手机操作系统经常无法让用户有效控制并了解第三方应用程序对隐私敏感性数据的使用方式。我们构建了一个高效的全系统动态污点跟踪和分析系统TaintDroid，用来解决上述弊端。它能够同时跟踪多个敏感数据来源。TaintDroid 利用 Android 的虚拟化执行环境来提供实时分析。通过对 30 款受欢迎的第三方 Android 应用程序的行为进行监控，我们在其中的 20 款中共发现了 68 例滥用用户位置信息和设备识别信息的行为。利用 TaintDroid 监控敏感数据能让手机用户了解第三方应用程序的使用行为，并为寻求识别不良应用程序的智能手机安全服务企业提供宝贵信息。

1.引言

现代智能手机平台的一大特点在于使用集中式服务下载第三方应用程序。这类“应用程序商店”为用户和开发者带来了便利，提高了移动设备的娱乐性和实用性，并且促进了应用程序开发的蓬勃发展。其中众多应用程序都将远程云服务的数据与本地传感器的信息相结合，这些本地传感器包括 GPS 接收器、摄像头、麦克风以及加速计等。应用程序通常具有访问这类隐私敏感性数据的合理原因，但是用户却希望确保其数据被正当使用。

智能手机平台所面临的一项严峻挑战就是，如何解决第三方应用程序提供的娱乐性和实用性与其带来的隐私安全隐患之间的矛盾。目前，智能手机操作系统仅提供粗粒度的控制来管理应用程序是否能够*访问*隐私信息，却很少提供关于隐私信息实际使用方式的信息。比如，用户允许某应用程序访问其位置信息后，他/她无从知晓该应用程序会将此信息发送给基于位置的服务，还是广告发布者，还是应用程序开发者或任何其他机构。因此，用户只能盲目地相信应用程序会妥善处理他们的隐私数据。

这一问题的固有性质决定了它无法通过传统的访问控制技术来解决。一种简单的解决方案是在某进程接收隐私敏感性信息时关闭网络访问。然而，与其他平台相比，智能手机应用程序或许更依赖云服务。这带来了两个重要结果。首先，绝大多数应用程序的运行都需要访问网络。其次，更重要的一点是，一些应用程序必须向特定网络主机发送隐私敏感性信息才能满足用户的需求。因此，这一问题不单在于确定这类信息*是否*被发送至网络，更在于确定被发送信息的*内容*和*去向*。

要想确定应用程序如何使用并披露隐私敏感性信息，可以使用细粒度动态污点分析，即众所周知的“污点跟踪”。“污点”是位于数据项或变量中的标签。该标签将某个语义类型（如地理位置）分配给数据，并可同时编码多个语义类型（一般称为*污点标记*）。污点跟踪系统的任务是 (1) 在*污点来源{2}上分配污点标签*，*(2) 自动将污点标签传播给依赖数据和变量*，*(3) 根据污点流动终点*中的数据污点标签采取某些操作。比如，当包含手机地理坐标的变量被位置 API（污点来源）返回时，污点跟踪系统可能会将其进行标记，并将该标签传播给由此类变量衍生而来的所有变量（比如，假设`a = b + c`，那么`a`由`b`和`c`衍生而来}，然后在包含位置标签的变量到达网络 API（污点流动终点）时采取某些操作（记录和丢弃）。

安全领域的文献中包含很多污点跟踪的实例，但提出的解决方案或者粗略，或者低效。我们展示了 Android 应用程序是可以高效地进行污点跟踪的。此外，我们发现，仅监控单个进程对 Android 而言是不够的，因为数据一般会在应用程序之间流动。

我们的目标是创建一个能够*实时*运行的全系统污点跟踪框架，以便在充足的上下文中检测敏感数据泄露，从而发现潜在的不良应用程序。实时约束既满足了相关用户的日常使用需要，也让外部安全服务得以

本文的最初版本刊登在 *2010 年《第九届 USENIX 操作系统设计和实现专题研讨会论文集》*中。

进行有效研究。正如其他同类实用系统，该解决方案在设计上必须对性能和精度做出仔细权衡。

我们介绍的 TaintDroid 是基于 Android 的扩展，它是首个通过智能手机平台来跟踪隐私敏感性数据流的实用系统。为平衡性能与跟踪精度，TaintDroid 利用 Android 的虚拟化架构整合了四种粒度的污点传播，即变量级别、方法级别、消息级别以及文件级别。尽管这些并不是新技术，但我们的贡献在于对这些技术加以整合，并针对资源受限的智能手机找到了跟踪精度与性能之间的适当平衡点。

我们的另一个贡献在于使用 TaintDroid 开展了首个同类智能手机应用程序研究，旨在识别大量滥用隐私敏感性数据的行为。这项研究涉及到 30 款随机选取的热门 Android 应用程序，而且它们均会用到位置、摄像头或麦克风数据。我们在安装 TaintDroid 固件的手机上手动运行这些应用程序，然后收集各种 TaintDroid 日志和网络日志，并记录用户关于隐私敏感性数据泄露的预期，以便评估潜在的数据滥用行为。在我们的实验中，TaintDroid 准确地标记了 105 个包含隐私敏感性信息的 TCP 连接。经进一步检查，其中的 37 个连接被明确列为合法连接。通过检查其余 68 个 TCP 连接，我们发现，在这 30 款应用程序中有 15 款向广告服务器发送了用户的位置信息；共有 7 款收集了设备 ID 信息，甚至在有些情况下，它们还收集了电话号码和 SIM 卡序列号信息。总体而言，本研究中三分之二的应用程序以可疑方式使用了敏感数据。这些研究结果让人们更加担忧，应用程序可能会在用户毫不知情的情况下广泛收集地理位置信息和手机识别信息。

2.设计概况

我们所寻求的设计方案应该能让用户实时监控智能手机上的第三方应用程序如何实时处理隐私数据。要求源代码的现有静态分析技术并不适合，因为许多智能手机应用程序是闭源的，而且用于将字节码转化为源代码的技术还远远达不到零错误。即便源代码可用，信息的使用方式也往往由运行时事件和配置决定；实时监控可以说明这些特定于环境的依赖关系。此外，我们假设所有下载的第三方应用程序都是不受信任的，而且这些应用程序同时运行。

监控智能手机上的隐私敏感性信息的网络泄露情况时，面临着几大挑战：

- *智能手机的资源是受约束的。*由于智能手机的资源受到限制，这就妨碍了重量级资源跟踪系统的使用。
- *第三方应用程序被托管了多种隐私敏感性信息。*监控系统必须区别多种信息类型，这需要额外的计算和存储。
- *即使以明文发送的隐私敏感性信息仍难以识别。*比如，地理位置信息属于频繁更改且难以预测的浮点数对。
- 应用程序可以共享信息。如果将监控系统局限在单个应用程序，就无法追踪通过文件和进程间通信 (IPC)传递的数据流，其中包括用于传播隐私敏感性信息的核心系统应用程序。

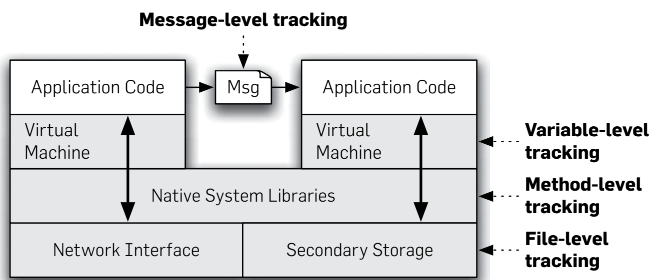
全系统细粒度动态污点分析能够化解上述挑战，但前提是解决性能约束和过度(污点)跟踪的问题。在这里，敏感信息最先在*污点来源*中进行识别并被分配一个表示信息类型的污点标志。智能手机拥有明确定义的应用程序编程接口 (API)，用于检索隐私敏感性信息（如麦克风数据、位置信息以及手机识别信息等）。随后，动态污点分析技术跟踪带标签的数据如何以可能泄露原始敏感信息的方式来影响其他数据。此跟踪通常在指令级别执行。比如，如果执行`a = b + c`的指令，而`c`带有污点 t，那么该指令执行后，`a`将带有污点 t。最后，受影响的数据在离开系统前会在*污点流动终点*（我们的设计方案中的网络接口）中进行识别。

显然，在指令级别执行动态污点分析会造成巨大的性能开销。比如，全系统仿真和每进程动态二进制转换 (DBT)[2, 4, 17] 一般会造成 2 至 20 倍的性能下降。这一开销的产生原因是，对每个受监控的指令而言，跟踪框架必须 (1) 保存执行上下文，(2) 执行污点传播，然后 (3) 恢复执行上下文。

我们克服该缺陷的方法是，将该跟踪框架移入操作系统内部并充分利用 Android、BlackBerry 和 Windows Phone 所采用的基于虚拟机 (VM) 的架构。在上述平台中，应用程序由在解释器中执行的 Java 字节码或 .NET 字节码构成。我们修改解释器来执行污点传播，然后仔细地将传播扩散至系统的其余部分。通过修改解释器，我们就不再需要保存和恢复执行上下文了。此外，我们的方法侧重于跟踪由解释器代码使用的数据，它们在整个进程内存中所占比例非常小。值得注意的是，我们的方法并不适用于 iOS，因为该平台使用二进制应用程序。

图 1 展示了我们的跟踪设计方案。跟踪以四种方式进行。第一，我们检测 VM 解释器以便在不受信任的应用程序代码中进行*变量级别跟踪*。使用变量语义可以提供优于传统 x86 跟踪逻辑的精度，并且更加侧重于在数据上而不是代码上存储污点标志。第二，我们在应用程序之间使用*消息级别跟踪*。消息粒度能够将 IPC 开销降至最低，同时将分析扩展至整个系统范围。第三，对于系统提供的原生库，我们使用*方法级别跟踪*。这里，我们运行不经插桩的平台原生代码，并将污点传播赋予返回值。最后，我们使用*文件级别跟踪*以确保永久性信息能够以保守的方式保留其污点标志。

图 1.运用多级别方法在 Android 中进行高性能污点跟踪。



虽然这种设计实现了颇具实用性的跟踪，但它却依赖于固件的完整性。我们信任在用户空间执行的虚拟机以及由不受信任的解释型应用程序加载的任何原生系统库。因此，我们假设只有平台原生库可以加载。否则，应用程序不仅能够清除污点标志，还可以破坏解释器内的跟踪。我们在目标平台 (Android) 修改了原生库加载程序，使其仅加载来自固件的原生库。为测试兼容性，我们在 2010 年 7 月对总共 1100 款应用程序进行了调研，它们均是 Andorid Market 上各类别中排名前 50 位的最受欢迎的免费应用程序，结果表明：包含 .so 文件的应用程序所占比例不足 5%。因此，我们预期TaintDroid 仅与一小部分应用程序不兼容。

3.TAINTDROID

TaintDroid 实现了在 Android 平台上的多粒度污点跟踪。设计的关键在于对跟踪精度和性能进行仔细的权衡取舍。TaintDroid 在 VM 解释器中使用变量级别跟踪。多个污点标志被存储为一个污点标记。当应用程序执行原生方法时，变量污点标记会赋予其返回值。最后，污点传播扩展至 IPC 和文件。

本节概述 TaintDroid 在实现中面临的主要挑战。此处我们将探讨 (a) 污点标记存储，(b) 解释型代码污点传播，(c) 原生代码污点传播，(d) IPC 污点传播，以及 (e) 二级存储污点传播。更多详细内容可参见我们的最初论文。[9]

3.1.污点标记存储

污点标记的存储会同时影响性能和内存开销。传统的污点跟踪系统为每个数据字节或字存储一个标记。[3, 23] 通常在实现中该标记包含单独一个比特位。为进一步降低存储开销，这类系统使用非相邻影子内存[23]或标记图[25]，仅维护污染字节的标记。TaintDroid 则采取截然不同的方法。由于我们知道哪些字节是变量，我们可以通过仅持续跟踪变量的污染状态来显著缩小要跟踪的内存范围。这样一来，TaintDroid 就能够将污点标记存储在与变量相邻的内存中，这将为污点标记访问提供良好的空间局部性。另外，它允许对每个变量实际存储一个 32 位的位向量，从而实现了 32 种不同的污点标志。

TaintDroid 为 Android 的 Dalvik VM 转换器中的所有标量值都添加了污点标记存储。Android 的应用程序以 Java 编写，却编译成由 Dalvik 执行的特殊 DEX 字节码。考虑到与 Java 的这些渊源，TaitDroid 必须为方法局部变量、方法参数、类静态字段、类实例字段以及数组提供污点标记存储。

DEX 字节码与 Java 字节码的区别在于前者基于寄存器。这对 TaintDroid 的实现是非常重要的。当 DEX 方法被调用时，Dalvik 会创建一个新的堆栈帧，用于为该方法所用的所有标量和对象引用变量分配 32 位寄存器存储。如图 2 所示，方法参数也存储在堆栈上，并被映射至被调用方堆栈帧中的高指数寄存器。TaintDroid 通过在寄存器之间交错污点标记来为这些变量提供污点标记存储。

TaintDroid 将与类字段和数组相邻的污点标记存储在内部数据结构中。每个数组仅使用一个污点标记，以便将存储开销降至最低，这对字符串来说往往已经足够了。然而，精度上的这种损失可能导致误报。比如，当受污染的值被存储在数组中时，从该数组读出的所有值都会受到污染。幸运的是，Java 数组常常包含不易受污染的对象引用，从而减少了实际中的误报。

3.2.解释型代码污点传播

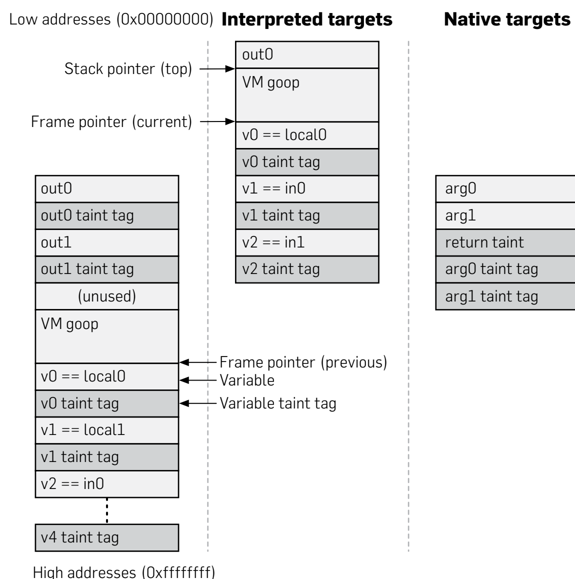
DEX 字节码运算为 TaintDroid 带来了多项明显优势。首先，所有的运算都有明确的语义。与 x86 不同的是，并不缺乏用于清除变量（如，`xor %eax-, %eax`）的寄存器或特殊约定。其次，标量值不同于指针。这使得污点传播更加精确。最后，非方法局部变量带有可保留类型的明确污点标记存储（如上所述）。

在大多数情况下，污点标记传播正如人们所预料的那样进行。指令始终会覆盖目标寄存器；因此，一元运算将目标寄存器的污点标记设置为源寄存器的污点标记，二元运算（如 `a = b + c`）将目标寄存器的污点标记设置为两个源寄存器的污点标记的并集（如 $\tau(a) \leftarrow \tau(b) \cup \tau(c)$）。就实现而言，该并集仅仅是污点标记位向量的一个位 OR。但是，在一些情况下，污点传播并不是直接的（如数组下标和对象引用）。完整的传播逻辑及相关探讨可参见我们的最初论文。[9]

3.3.原生代码污点传播

原生代码在 TaintDroid 中不受监控，因为执行自动污点传播需要重量级技术，如动态二进制转换 (DBT)。事实上，我们会在该方法终止后，将源代码检查和简单启发相结合来合成污染状态。

图 2. 修改后的堆栈格式。污点标记在解释型方法目标寄存器和原生方法附加寄存器之间交错排列。暗灰色方框代表污点标记。



内部 VM 方法。Dalvik VM 包含一套核心方法，它们可由解释型代码直接调用，同时传递一指向 32 位寄存器参数数组的指针，以及一个指向返回值的指针。TaintDroid 将所有污点标记放置在参数值之后（不妨回忆一下图 2 中的堆栈）。这样一来，只要方法不影响污点传播，就无需修改。而对于那些影响污点传播的方法，可以方便地使用相应的污点标记。在 Andorid 2.1 版本的 185 个内部 VM 方法中，只有 5 个方法需要修补（如，数组处理和反射）。

JNI 方法。其余原生方法大多使用 Java 原生接口 (JNI) 并通过 JNI 调用桥来调用。调用桥解析 Java 参数并分配一个返回值，因此它是在执行原生方法后修补跟踪状态的理想位置。为此，我们定义了一个方法配置文件表，其中定义了一系列 (*from*, *to*) 可指示方法参数、类变量和返回值之间的流动的对。使用自动静态分析工具来完全填充方法配置文件表的效果最佳；但是，为满足本研究的目的，我们根据需要人工定义了几种方法。为补充该人工制定的规范，我们创建了一个传播启发：*将方法参数污点标记的并集分配至返回值的污点标记*。如果该方法只采用原始参数、字符串参数以及返回值，那么这种启发是保守的。对于 Android 2.1 版本，我们发现 2844 种 JNI 方法中的 913 种属于这种情况。其余方法可能存在漏报，并且可能需要明确的方法配置文件规范。尽管我们发现，这些方法对我们的调查行之有效，但对原生代码进行更全面的思考是今后研究的重要方向。

3.4.IPC 污点传播

当 Andorid 应用程序彼此通信时，它们会通过绑定 IPC 接口发送包对象。对 TaintDroid 而言，有必要通过包来传播污点标记，从而跟踪在已下载第三方应用程序之间传递的敏感信息，以及在第三方应用程序与系统之间传递的敏感信息。实际上，Android 的多数核心功能都是使用与第三方软件相同的应用程序抽象来实现的。

TaintDroid 为每个包消息分配一个污点标记。与包中的变量级别或字节级别跟踪相比，这种做法实现的性能更高，内存开销更低。此外，变量级别跟踪可被操控，因为对不同大小的变量进行的打包是由发送方和接收方定义的。但是，其缺陷在于误报（与数组相似）。正如第 7 节所述，这会让某些污点来源成为 TaintDroid 的问题。未来的实现将研究细粒度包跟踪的开销。

3.5.次级存储污点传播

TaintDroid 必须确保，当受污染的数据写入文件时，污点标记会在该文件稍后被读取时恢复。我们目前为每个文件存储一个污点标记，因为细粒度跟踪会导致重大开销。但是，当被跟踪信息的类型频繁混合时，误报就成为了这种方法的弊端。在我们的实验中，这并不是一个严重的问题。为存储污点标记，TaintDroid 使用了文件系统中的扩展属性。当 TaintDroid 开发出来的时候，最广泛使用 YAFFS2 文件系统并不带有 xattr 支持，因此我们必须添加这种支持。YAFFS2 后来加入了正式的 xattr 支持，而新推出的手机还配备了支持标准 ext4 文件系统的硬件闪存转换层。Android 存储架构的另一个局限在于 SD 卡。Android 在 SD 卡上使用 FAT 文件系统，该系统并不支持 xattr。我们将 SD 卡格式化为 ext2 格式并修补了文件写入 API，以便使用与 FAT 相一致的文件权限，从而确保与现有应用程序兼容。

4.隐私HOOK的放置

要使用 TaintDroid 来监控应用程序，必须将污点来源添加至 Android 框架。我们修改了 Android 系统代码，以便将污点标记添加至各种污点来源。在大多数情况下，我们选择将污点来源添加至系统应用程序的 Java 部分中，这些应用程序可检索来自硬件的值。下文介绍了我们所遇见的最重要的污点来源类别。

低带宽传感器。许多类型的隐私敏感性信息是通过低带宽传感器（位置和加速计等）获取的。这类信息往往变化频繁，并且由多个应用程序同时使用。因此，Android 使用传感器管理工具来以多路复用方式访问低带宽传感器。这种传感器管理工具是放置污点来源 hook 的理想点。我们将 hook 放置在 Android 的 LocationManager 和 SensorManager 应用程序中。

高带宽传感器。麦克风和摄像头等来源都属于高带宽。来自传感器的每个请求都会返回大量仅能被某个应用程序所使用的数据。因此，操作系统通过大型数据缓冲区或文件（或二者兼具）来获取传感器信息。当传感器信息通过文件共享时，该文件必须使用相应的标记进行污染。对于为访问麦克风和摄像头接口而提供的两种 API 抽象，我们都添加了 hook。

信息数据库。通讯簿和短信等共享信息通常存储在基于文件的数据库中。通过向这类数据库文件添加污点标记，所有从文件读取的信息都将自动被污染。我们最初使用这种技术来跟踪通讯簿信息。后来的实现方案修改了 Android 的内容解析程序类，以根据查询应用程序所指定的内容提供程序的名称（如“权限字符串”）来添加恰当的污点标记。

设备标识符。手机或用户的唯一识别信息属于隐私敏感性信息。并不是所有的个人识别信息都易于污点跟踪。但是，手机包含一些易于污染的标识符：手机号码、SIM 卡标识符 (IMSI, ICC-ID) 以及设备标识符 (IMEI) 都可以通过定义完好的 API 访问。我们检测了 API 以获取手机号码、ICC-ID 以及 IMEI。IMSI 污点来源的固有局限性已在第 7 节中进行了讨论。

基于网络的污点流动终点。TaintDroid 可识别受污染信息何时被传输出网络接口。我们基于解释器的方法要求 TaintDroid 的代码在解释型代码中检测网络传输。因此，我们在Java 框架库中调用原生套接字库的时间点上进行了插桩。

5.应用程序研究

为证明 TaintDroid 的实用性，我们研究了 30 款流行的第三方 Android 应用程序，它们均可以访问用户的隐私敏感性数据和互联网。这组应用程序是从范围更大的一组流行应用程序中随机选出的，后者均可以访问互联网，并且至少可以访问位置、摄像头或音频数据中的一种。在随机选取应用程序时，我们更侧重于那些可以访问有趣的隐私敏感性信息的应用程序，因为无访问权限的应用程序显然无法泄露数据。关于我们的实验方法的详细内容，可参见我们的最初论文。[9]下文将阐述我们的主要发现。

我们的实验包括手动运行和研究上述应用程序的功能。我们记录了 TaintDroid 日志和一个 tcpdump 数据包追踪，以便获取真实数据。我们也记录了最终用户许可协议 (EULA) 和关于数据泄露的隐含预期。我们的实验共生成了 1130 个 TCP 连接，TaintDroid 正确地标记了 105 个含有受污染隐私敏感性信息的连接（也就是说，TaintDroid 没有出现误报）。标记的 TCP 连接包括纯文本和二进制编码数据。

我们对 105 个被标记为含有隐私敏感性信息的 TCP 连接进行了检查，发现其中的 37 个明确属于合法用途。例如，其中数个被标记的 TCP 连接均含有 HTTP 标头，指示使用了 Google Maps for Mobile (GMM) API，而相应的应用程序则显示了用户所在位置的地图。然而，其余 68 个被标记的 TCP 连接却泄露了隐私敏感性数据。表 1 汇总了这些调查结果。

发送到广告服务器的位置数据。半数接受研究的应用程序均在未获得用户明确或默许同意的情况下，向第三方广告服务器泄露了位置数据。在这 15 款应用程序中，只有两款在首次运行时显示了最终用户许可协议，然而，这两份协议并未提及上述泄露数据的行为。被泄露的位置信息兼有纯文本和二进制两种格式。后者凸显出了 TaintDroid 相对于基于模式的简单数据包扫描的优势。应用程序将位置数据以纯文本形式发送到 admob.com、ad.qwapi.com、ads.mobclix.com（11 款应用程序），以二进制格式发送到 FlurryAgent（4 款应用程序）。泄露给 AdMob 的纯文本位置数据显示在 HTTP GET 字符串中：

```
...& s=a14a4a93f1e4c68 &..& t=062A1CB1D-
476DE85B717D9195A6722A9&d%5Bcoor-
d%5D=47.661227890000006%2C−122.31589477&...
```

针对 AdMob SDK 的调查表明，s = 参数是某应用程序发行者独有的标识符，coord= 参数则提供了地理坐标。

对于由 FlurryAgent 发送的二进制数据，我们则根据以下一系列事件来确认位置数据遭到泄露。首先，名为“FlurryAgent”的组件通过向位置管理期进行注册来接收位置更新。然后，TaintDroid 日志消息显示了从位置管理器接收污染包的应用程序。最后，当应用程序收到污染包后，其 Android logcat 日志会立即报告：“正在向 http://data.flurry.com/aar.do 发送报告”。

我们的实验表明，这十五款应用程序收集了位置数据，并将其发送至广告服务器。在某些情况下，即使应用程序并未显示任何广告，位置数据依然被发送至广告服务器。但我们通过 TaintDroid 证实，三款接受研究的应用程序（不包含在表 1 中）只根据用户的请求传输位置信息，进而向服务器提取本地内容。这一发现表明了监控应用程序如何在*实际中*使用或滥用所授予权限的重要性。

手机信息。在 30 款接受研究的应用程序中，20 款要求获得读取手机状态并访问互联网的权限。我们发现，在这20款应用程序中，有2款向其服务器传输了：(1)设备的手机号码、(2) IMSI，即用于识别 GSM 网络中的具体用户的唯一15位代码，以及(3) ICC-ID 号码，即唯一 SIM 卡序列号。我们通过检查信息的纯文本来验证标记是否正确。两款应用程序都没有告知用户将通过手机发送这些信息。请注意，虽然我们没有显式跟踪 IMSI（见第7节），但它包含在由TaintDroid标记的纯文本网络缓冲区中。

表 1。应用程序研究结果。

应用程序 [package.name]	权限				信息发送		
	位置	手机状态	摄像头	麦克风	位置	电话信息	IMEI
Babble Book [com.kalicinscy.babble]	✓						
Cestos Full [com.chickenbrickstudios.cestos_full]	✓						
Manga Browser [com.mangabrowser.main]	✓				•		
Movies and showtimes [com.stylem.movies]	✓						
Solitare Free [com.mediafill.solitaire]	✓				•		
The Weather Channel [com.weather.Weather]	✓						
3001 Wisdom Quotes Lite [com.xim.wq_lite]	✓	✓			•		
Antivirus Free [com.antivirus]	✓	✓			•	•	•
Astrid [com.timsu.astrid]	✓	✓			•		
BBC News listen & tweet [daaps.media.bbc]	✓	✓			•		
Blackjack [spr.casino]	✓	✓			•		
Bump [com.bumptech.bumpga]	✓	✓					•
Children’s ABC Animals (lite) [com.mob4.childrenabc.animals]	✓	✓			•		
Hearts (Free) [com.bytesequencing.hearts_ads]	✓	✓			•		
Horoscope [fr.telemaque.horoscope]	✓	✓			•		•
Mabilo Ringtones [mabilo.ringtones]	✓	✓			•		
The directory for Germany [de.dastelefonbuch.android]	✓	✓					
Traffic Jam Free [com.jiuzhangtech.rushhour]	✓	✓			•		
Wertago for Nightlife [com.wertago]	✓	✓			•		•†
Yellow Pages [com.avantar.yp]	✓	✓					•
Knocking Live Video Beta [com.pointyheadsllc.knockingvideo]	✓	✓	✓				•
Layar [com.layar]	✓	✓	✓				
Pro Basketball Scores [com.plusmo.probasketballscores]	✓	✓	✓		•		○
Slide:Spongebob [com.mob4.slideme.qw.android.spongebob]	✓	✓	✓		•		
The coupons App [thecouponsapp.coupon]	✓	✓	✓			•	•
Trapster [com.trapster.android]	✓	✓	✓				•
Barcode Scanner [com.google.zxing.client.android]			✓				
iXmat Barcode Scanner [com.ixellence.ixmat.android.community]			✓				
Myspace [com.myspace.android]			✓				
Evernote [com.evernote]	✓		✓	✓			

✓ = 潜在违规；○= 在最终用户许可协议中明确声明；† 发送哈希值。

这一发现表明，Android 的粗粒度访问控制未能提供充足的保护来防范试图收集敏感数据的第三方应用程序。此外，我们发现一款应用程序在手机每次启动时都会传输手机信息。尽管该应用程序在首次使用时显示了使用条款，但其中却并未就收集此类高度敏感信息做出说明。出人意料的是，该应用程序在安装完成后，甚至还没有等到用户使用，就立刻传输手机数据。

设备唯一 ID。设备的 IMEI 同样遭到应用程序的泄露。IMEI 可对特定手机进行唯一识别，用于防止被盗手机访问蜂窝网络。TaintDroid 的标记表明，9 款应用程序传输了 IMEI，其中的 7 款既没有显示最终用户使用协议，也没有在其中就收集 IMEI 作出说明。7 款应用程序中有一款是流行的社交网络应用程序，还有一款是基于位置的搜索应用程序。此外，我们发现 7 款应用程序有 2 款在向其内容服务器传输设备的地理坐标时包含了 IMEI，从而有可能将 IMEI 改用作客户端 ID。

相比之下，9 款应用程序中有 2 款更加谨慎地对待 IMEI。其中一款应用程序在其隐私声明中明确表示会收集设备 ID，另一款应用程序则使用了 IMEI 的哈希值而非数值本身。我们通过比较两部不同手机的结果证实了这一行为。使用 IMEI 的哈希值可以提供更多保护，因为它无法倒回真实的 IMEI。但是，如果所有应用程序都直接将 IMEI 转换成哈希值，就会引起类似的隐私担忧。

第三方应用程序可以通过收集手机标识符来跟踪用户的行为。手机号码往往可以很容易地与机主姓名关联起来，因为许多用户将手机号码公布在社交网络和其他网站上。另外，即使是收集看似无法辨识的号码，如 IMSI、ICC-ID 和 IMEI 等，也会对用户的隐私权造成影响。第一，手机上的所有应用程序都使用相同的手机标识符。如果标识符和行为被某个与很多应用程序相关联的机构（例如，广告或分析服务）所收集，就可以创建更加准确的用户资料。第二，这些标识符在用户使用手机期间是固定不变的，而且，如果 SIM 卡被安装到新的手机中，则标识符可能在更长时期内保持不变。这意味着用户无法像使用 Web 浏览器那样简单地清除跟踪 cookie。最后，这些标识符通常会连同电子邮件地址等个人识别信息一起被收集。这些收集起来的数据可以用于创建小型数据库，从而将真实用户与其手机标识符相关联。过去，这种映射信息仅仅由手机运营商保存。

表 2。宏基准测试的结果。

	Android (ms)	TaintDroid (ms)
应用程序加载时间	63	65
通讯簿（创建）	348	367
通讯簿（读取）	101	119
打电话	96	106
拍照片	1718	1718

6.性能评估

在应用程序研究中，TaintDroid只引入了很小的性能开销。原因是：(1) 大多数应用程序主要处在“等待状态”，(2) 重量级运算（例如，屏幕更新和网页渲染）均发生在不受监控的原生库中。

为了评估适用于 Android 2.1 版本的 TaintDroid 的性能，我们使用了宏基准测试来表示智能手机的常见活动：加载应用程序、访问通讯簿、打电话和拍照片。如表 2 所示，宏观基准测试仅观察到微不足道的开销（少于 30 毫秒），但拍照片除外，该活动用时超过半秒。造成这种开销的原因可能是当前方法将污点标记传播到使用 xattrs 的文件，而通过缓存可能会改善这一情况。

我们不仅通过宏基准测试报告了用户在使用智能手机常见功能时可感知到的性能开销，同时还针对 Java 运算执行了微基准测试。在该实验中，我们针对 Java 使用了标准 CaffeineMark 3.0 基准测试的 Android 端口。TaintDroid 的总体平均 CPU 开销为 14%。我们还在实验中衡量了基准测试进程的内存消耗。基准测试进程在 Android 和 TaintDroid 上分别耗用了 21.28 MB 和 22.21 MB，这表明内存开销为 4.4%。

7.探讨

方法的局限性。为了将性能开销降至最低，TaintDroid 仅跟踪数据流（即显式流），而不跟踪控制流（即隐式流）。从第 5 节中可以看出，TaintDroid 可以跟踪敏感数据流，并识别泄露敏感信息的众多应用程序。然而，真正的恶意应用程序可以躲过我们的系统，并通过控制流来泄露隐私敏感性信息。完全跟踪控制流需要静态分析，[7, 14]这是对第三方应用程序来说是难以付诸实践的，因为它们的源代码不可用。如果污点范围可以确定，就能动态跟踪直接控制流[21]；然而，DEX 并不维护可供 TaintDroid 利用的分支结构。用于确定方法控制流图 (CFG) 的按需静态分析提供了这一上下文[15]；然而，为避免误报和重大性能开销，TaintDroid 目前并不执行这种分析。我们的数据流污点传播逻辑与现有的著名污点跟踪系统是一致的。[3, 23]最后，一旦信息离开手机，它可能以网络应答的形式返回。当信息离开手机后，TaintDroid 就无法跟踪此类信息传播了。

实现方案的局限性。Android 使用了 Java 的 Apache Harmony 实现方案，并进行了一些自定义修改。该实现方案纳入了对 *PlatformAddress* 类的支持，PlatformAddress 类包含一个原生地址，并由 *DirectBuffer* 对象使用。文件和网络 IO API 包括了写和读的“直接”变体，这些变体使用了来自 *DirectBuffer* 的原生地址。TaintDroid 目前并不跟踪 *DirectBuffer* 对象上的污点标记，因为该数据存储在不透明的原生数据结构中。目前，TaintDroid 能在读或写的“直接”变体被使用时进行记录，但这种情形发生的频率极低。对于通过原生地址运行的 *sun.misc.Unsafe* 类，在实现方案上也存在类似的局限。

污点来源的局限性。虽然 TaintDroid 可以有效地跟踪敏感信息，但当被跟踪的信息包含配置标识符时，它会产生重大误报。例如，IMSI 数值字符串由移动国家代码 (MCC)、移动网络代码 (MNC) 和移动站识别码 (MSIN) 组成，所有这些代码都会一同受到污染。Android 在传送其他数据时广泛使用 MCC 和 MNC 作为配置参数。如果 IMSI 被视为受到污染，会导致同一包中的所有信息都受到污染，最终造成受污染信息激增。因此，对于包含配置参数的污点来源，对包内的个别变量进行污染更为合适。然而，正如第 5 节的分析结果所示，消息级别的污点跟踪对绝大部分污点来源都是行之有效的。

8.相关研究

在过去的几十年中，信息流跟踪和控制一直是众多操作系统和编程语言设计的基础。为简略起见，我们侧重于使用动态污点分析技术的系统，该分析技术主要用于跟踪旧版程序中的信息流。我们已使用该技术来提高系统完整性（如防御软件攻击[4, 16, 17]）和保密性（如发现隐私泄露[8, 23, 25]），同时跟踪互联网蠕虫。[5] 动态跟踪方法既包括使用硬件扩展[6, 19, 20]和仿真环境[3, 23]进行的全系统分析，也包括使用动态二进制转换 (DBT)[2, 4, 17, 25] 进行的每进程跟踪。TaintDroid 的设计灵感源于这些早期研究，但却解决了移动电话所特有的难题。据我们所知，TaintDroid 是第一款手机专用的污点跟踪系统，也是首个通过整合数据对象多粒度跟踪来实现实用全系统分析的动态污点分析系统。

最后，动态污点分析已应用于虚拟机和解释器。Haldar 等人[10] 利用污点跟踪检测 Java 字符串类以防范 SQL 注入攻击。WASP[11] 的动机与之类似；但它对个别字符进行积极污染，以确保 SQL 查询仅包含高完整性的子字符串。Chandra 和 Franz[1] 提出在 JVM 中使用细粒度信息流跟踪，并检测 Java 字节码以协助进行控制流分析。与之类似，Nair 等人[15] 检测了 Kaffe JVM。Vogt 等人[21] 对 Javascript 解释器进行检测以防范跨站脚本攻击。Xu 等人[22] 使用动态信息跟踪自动检

测 PHP 解释器源代码，以防范 SQL 注入攻击。最后，PHP 和 Python 的 Resin[24] 环境使用数据流跟踪来防范各种 Web 应用程序攻击。当数据离开解释型环境时，Resin 会实现文件过滤器和 SQL 数据库，以字节级粒度对对象和策略进行序列化和反序列化。TaintDroid 的解释型代码污点传播与上述研究存在相似之处。然而，TaintDroid 实现全系统信息流跟踪，将解释器污点跟踪与一系列操作系统共享机制无缝连接。

9.结论

虽然智能手机操作系统允许用户控制应用程序对敏感信息的访问，但用户难以了解应用程序如何使用他们的隐私数据。为解决此问题，我们推出了 TaintDroid，它是一个高效的全系统信息流跟踪工具，可以同时跟踪多个来源的敏感数据。TaintDroid 的一大设计目标就是高效。我们通过整合四个粒度（变量级别、消息级别、方法级别以及文件级别）的污点传播实现了这一目标。我们的评估显示，TaintDroid 在 CPU 密集型微基准测试中的性能开销仅为 14%。此前，关于污点跟踪的大多数研究工作或者效率低下（需要几倍的性能开销），或者需要源代码。由于 Android 应用程序的源代码不可用，人们可能因此认为 TaintDroid 的运行很慢。但事实并非如此：TaintDroid 无需源代码即可跟踪 Android 应用程序的信息流，而且开销适度。

我们使用 TaintDroid 对 30 款热门第三方应用程序的行为进行了研究，并发现其中三分之二的应用程序以不当方式处理敏感数据。特别值得一提的是，在这 30 款应用程序中，有 15 款与远程广告和分析服务器共享用户的位置信息。我们的研究结果证明，使用 TaintDroid 等监控工具来改善智能手机平台不仅富有成效，而且意义重大。

TaintDroid 是一项仍在开展的工作，它已被纳入本文作者和研究界其他人士今后将要开展的项目中。TaintDroid 适用于 Android 2.1 版本、2.3 版本（添加 JIT 支持）以及 4.1 版本。关于下载和构建 TaintDroid 的信息，请参见 http://www.appanalysis.org。

鸣谢

我们感谢为最初论文[9] 提供帮助的每位人士。Enck 和 McDaniel 得到了美国国家科学基金会 (NSF) 资助项目 CNS-0905447、CNS-0721579 和 CNS-0643907 的部分支持。Cox 和 Gilbert 得到了美国国家科学基金会杰出青年教授奖 CNS-0747283 的部分支持。

参考资料

1. Chandra, D., Franz, M. Fine-grained information flow analysis and enforcement in a Java virtual machine.In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC)* (Dec. 2007).
2. Cheng, W., Zhao, Q., Yu, B., Hiroshige, S. TaintTrace: efficient flow tracing with dynamic binary rewriting.In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)* (Jun.2006), 749–754.
3. Chow, J., Pfaff, B., Garfinkel, T., Christopher, K., Rosenblum, M. Understanding data lifetime via whole system simulation.In *Proceedings of the 13th USENIX Security Symposium* (Aug. 2004).
4. Clause, J., Li, W., Orso, A. Dytan: a generic dynamic taint analysis framework.In *Proceedings of the 2007 International Symposium on Software Testing and Analysis* (2007), 196–206.
5. Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L., Barham, P. Vigilante: end-to-end containment of internet worms. In *Proceedings of the ACM Symposium on Operating Systems Principles* (Oct. 2005), 133–147.
6. Crandall, J.R., Chong, F.T.Minos: control data attack prevention orthogonal to memory model.In *Proceedings of the International Symposium on Microarchitecture* (Dec. 2004), 221–232.
7. Denning, D.E., Denning, P.J.Certification of Programs for Secure Information Flow.*Commun.ACM 20, 7* (Jul.1977).
8. Egele, M., Kruegel, C., Kirda, E., Yin, H., Song, D. Dyanmic spyware analysis. In *Proceedings of the USENIX Annual Technical Conference* (Jun.2007), 233–246.
9. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Oct. 2010).
10. Haldar, V., Chandra, D., Franz, M. Dynamic taint propagation for Java. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)* (Dec. 2005), 303–311.
11. Halfond, W.G., Orso, A., Manolios, P. WASP: protecting web applications using positive tainting and syntax-aware evaluation.*IEEE Trans.Softw.Eng.34,* 1 (2008), 65–81.
12. Ho, A., Fetterman, M., Clark, C., Warfield, A., Hand, S. Practical taint-based protection using demand emulation.In *Proceedings of the European Conference on Computer Systems (EuroSys)* (Apr. 2006), 29–41.
13. Lam, L.C., cker Chiueh, T. A general dynamic information flow tracking framework for security applications.In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)* (Dec. 2006), 463–472.
14. Myers, A.C. JFlow: practical mostly-static information flow control. In *Proceedings of the ACM Symposium on Principles of Programming Langauges (POPL)* (Jan. 1999).
15. Nair, S.K., Simpson, P.N., Crispo, B., Tanenbaum, A.S.A virtual machine based information flow control system for policy enforcement.In *The 1st International Workshop on Run Time Enforcement for Mobile and Distributed Systems (REM)* (2007).
16. Newsome, J., Song, D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software.In *Proceedings of the 12th Network and Distributed System Security Symposium (NDSS)* (2005).
17. Qin, F., Wang, C., Li, Z., seop Kim, H., Zhou, Y., Wu, Y. LIFT: a low-overhead practical information flow tracking system for detecting security attacks. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture* (2006), 135–148.
18. Saxena, P., Sekar, R., Puranik, V. Efficient fine-grained binary instrumentation with applications to taint-tracking.In *Proceedings of the IEEE/ACM symposium on Code Generation and Optimization (CGO)* (Apr. 2008), 74–83,.
19. Suh, G.E., Lee, J.W., Zhang, D., Devadas, S. Secure program execution via dynamic information flow tracking. In *Proceedings of the Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Oct. 2004), 85–96.
20. Vachharajani, N., Bridges, M.J., Chang, J., Rangan, R., Ottoni, G., Blome, J.A., Reis, G.A., Vachharajani, M., August, D.I. RIFLE: an architectural framework for user-centric information-flow security.In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture* (2004), 243–254.
21. Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C., Vigna, G. Cross-site scripting prevention with dynamic data tainting and static analysis.In *Proceedings of the 14th Network and Distributed System Security Symposium* (2007).
22. Xu, W., Bhatkar, S., Sekar, R. Taint-enhanced policy enforcement: a practical approach to defeat a wide range of attacks.In *Proceedings of the USENIX Security Symposium* (Aug. 2006), 121–136.
23. Yin, H., Song, D., Egele, M., Kruegel, C., Kirda, E. Panorama: capturing system-wide information flow for malware detection and analysis.In *Proceedings of the 14th ACM Conference on Computer and Communications Security* (2007), 116–127.
24. Yip, A., Wang, X., Zeldovich, N., Kaashoek, M.F.Improving application security with data flow assertions.In *Proceedings of the ACM Symposium on Operating Systems Principles* (Oct. 2009).
25. Zhu, D.Y., Jung, J., Song, D., Kohno, T., Wetherall, D. Tainteraser: protecting sensitive data leaks using application-level taint tracking.*Operating Sys.Rev. 45,* 1 (2011), 142–154.

William Enck (enck@cs.ncsu.edu)，计算机科学系，北卡罗莱纳州立大学。

Jaeyeon Jung (jjung@microsoft.com)，微软研究院。

Peter Gilbert 和 Landon P. Cox ([gilbert, lpcox]@cs.duke.edu)，计算机科学系，杜克大学。

Patrick McDaniel (mcdaniel@cse.psu.edu)， 计算机科学和工程系，宾夕法尼亚州大学。

Byung-Gon Chun (bgchun@snu.ac.kr)，首尔国立大学。

Anmol N.Sheth (anmol.sheth@technicolor.com)，Technicolor Research。

译文责任编辑：陈海波

版权归属于作者 / 所有者发行权已授予 ACM。$15.00