★CACM 中国版★

计算机协会通讯

CACM.ACM.ORG 2014年6月第57卷第6期

Leslie Lamport

ACM 图灵奖得主

数据与分析之外的世界

排中律的诅咒

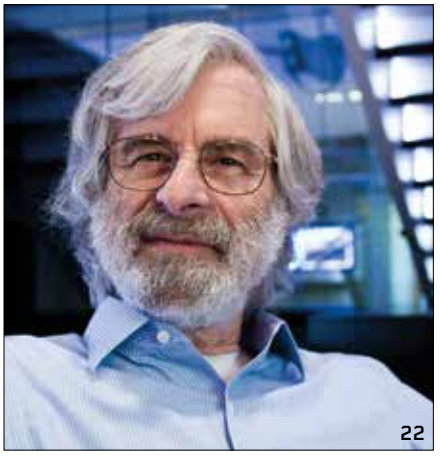
网络-物理测试床

动态共享内存

运动场

Association for Computing Machinery

新闻

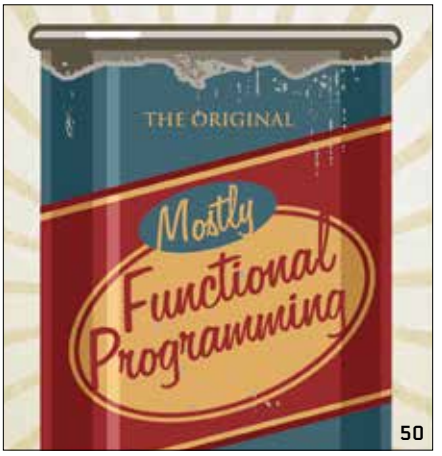


观点

关于封面：
Leslie Lamport, 2013 年ACM 图灵奖得主，照片摄于微软硅谷办公区，拍摄者为：位于旧金山的摄影师 Richard Morgenstein。Lamport 因其对分布式和并发系统理论与实践的基础性贡献而闻名。.

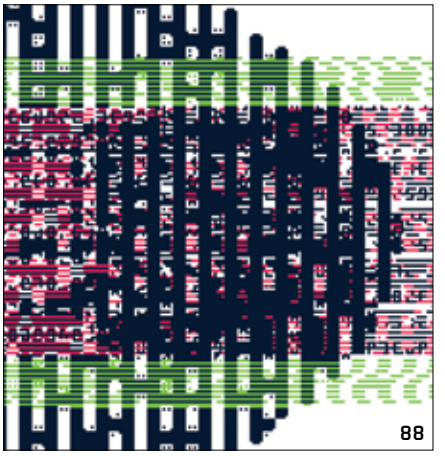
图片由 RICHARD MORGENSTEIN, ALICIA KUBISTA, IGNOTUS THE MAGE 提供

实践



投稿文章

评论文章



研究亮点

Association for Computing Machinery
Advancing Computing as a Science & Profession

里程碑 | DOI:10.1145/2601076 Neil Savage

拜占庭将军协议

分布式计算系统能够按照预期方式工作，Leslie Lamport所做的理论和实践研究功不可没。

当多个计算机处理器执行一些依赖对方输出的操作时，“哪一个发生在前”的问题就变得尤为重要。在阅读Paul Johnson和Robert Thomas 1975年发表的论文时，Leslie Lamport意识到了这个问题。在这篇论文中，Paul Johnson和Robert Thomas提出在分布式算法中加上时间戳来跟踪事件顺序。

“我立即联想到了狭义相对论（爱因斯坦的时空理论），”2013年图灵奖得主、微软研究院首席研究员Lamport表示。“‘同步’是一个相对的概念，并不是一成不变的。但是，因果关系的概念却是永恒不变的。”通过多台自主计算机相互通信，Leslie Lamport为这混乱的分布式计算系统过程建立了秩序，他将因此获得25万美元的图灵奖奖金。

根据爱因斯坦的理论，只有光在第二个事件发生之前，就从第一个事件到达此第二事件，两个事件才能构成因果关系。Lamport认为同样的概念也可以应用到计算机系统的信息传递活动中；如果想让第一个处理器影响第二个处理器，第一个处理器的输出就要在第二个处理器执行操作之前到达。要实现这一点，就必须对“之前”进行定义。

Lamport提出的逻辑时钟概念解决这一问题。逻辑时钟记录一系列点击事件，为每个事件分配一个逻辑时钟值。如果一个事件在另外一个事件之前发生，它的时钟值就一定小于后发生的事件。只要事件顺序确定，建立任意一个状态机就成为可能。“然后你就能实施想要的任何类型的系统，”Lamport说。

系统不发生错误时逻辑时钟运行良好，Lamport说；问题是如何让系统在出现错误时还能继续工作。1977年到1985年，Lamport就职于斯坦福国际研究院（SRI International）该研究院与美国国家航空航天局（NASA）签约，为其设计高可靠性的飞行器控制系统原型。假定一个具有三个处理器的系统，能容忍其中一个处理器发生故障：如果从某个传感器中获得输入，两个处理器计算出相似结果，而第三个处理器得出完全不同的结果，那么系统就会采用两个相似的结果，而抛弃第三个结果。

Lamport和斯坦福国际研究院的同事Robert Shostak与Marshall Pease在一篇论文中提出了这样一个问题：出错的那个处理器也许会向另外两个处理器发送不同的错误结果，处理器A和处理器B可能都认为出错处理器C的结果与自己相同，从而否定对方。该团队意识到，需要另外增加一个处理器避免发生此问题；事实上，允许n个处理器发生错误时系统还能正常工作需要3n+1个处理器。这篇论文获得了2005年分布式计算领域的Edsger W. Dijkstra奖。

Lamport将这个解决方案归功于他的同事们；他说自己的贡献是提出了使用数字签名的新算法，据此处理器就能辨别出谁的计算结果不一致，这样2n+1个处理器就能解决上述问题了。当时由于加密签名太过昂贵而难以实施，他的2n+1方案没有被采用。

他的功劳是使得这篇论文得以写完。Lamport说，“我写了第一稿，Shostak认为惨不忍睹，就完全重写了一遍。”

“‘同步’是一个相对的概念，这个概念并不是一成不变。但是，因果关系的概念却是永恒不变的。”

Shostak认为Lamport最大的贡献是在另外一篇论文中命名了这个问题，他在论文中把处理器描述为一群包围敌人营地的拜占庭将军，他们在一起决定是否发起攻击。出错的处理器就像是叛变的将军，告诉这个同事应该进攻，又告诉另外一个不应该进攻。这个方案现在叫做“拜占庭协议”。

“在宣传我们的工作成果方面，这起到了重要作用，”Shostak说，“Lamport不仅是一个拥有大量创新成果的显赫人物，还非常擅长推广计算机科学的研究成果”。

Lamport说，这种推广方式对Paxos，一种专门为容错状态机处理非“拜占庭将军问题”而提出的算法，失效了。他曾用希腊Paxos岛上议会如何达成一致意见的故事来阐明这种新算法。Lamport说，尽管这个算法很实用，但这个故事对其推广未起到帮助。

康奈尔大学计算机科学家Fred Schneider认为，Lamport的过人之处在于他用“第一性原理”的方法来解决计算机科学问题，描述问题时只做很少的、其他人可能没注意到的假设。“他不接受教条或对问题的普遍描述，”Schneider说。

关于程序是否正确终止，Lamport重新定义了“正确性”，其中包含两个概念：“安全性”——未出现问题及“活性”——发生积极事件。他还开发了LaTeX，这是一套文档处理系统标准，用于出版一些科学领域的文档。“他每次出手都能捡到宝石。”Schneider说道。

Lamport建议那些想成为系统设计专家的人：要学会更好地思考。“学会更好思考的方式就是多用数学思维，”他说，“所以首先应该接受良好的数学教育。”

Neil Savage是一名科技作家，来自马萨诸塞州，洛威尔市。

译文责任编辑：李向阳

照片由 RICHARD MORGENSTEIN 提供

DOI:10.1145/2602326

Charles K. Davis

观点
数据与分析之外的世界

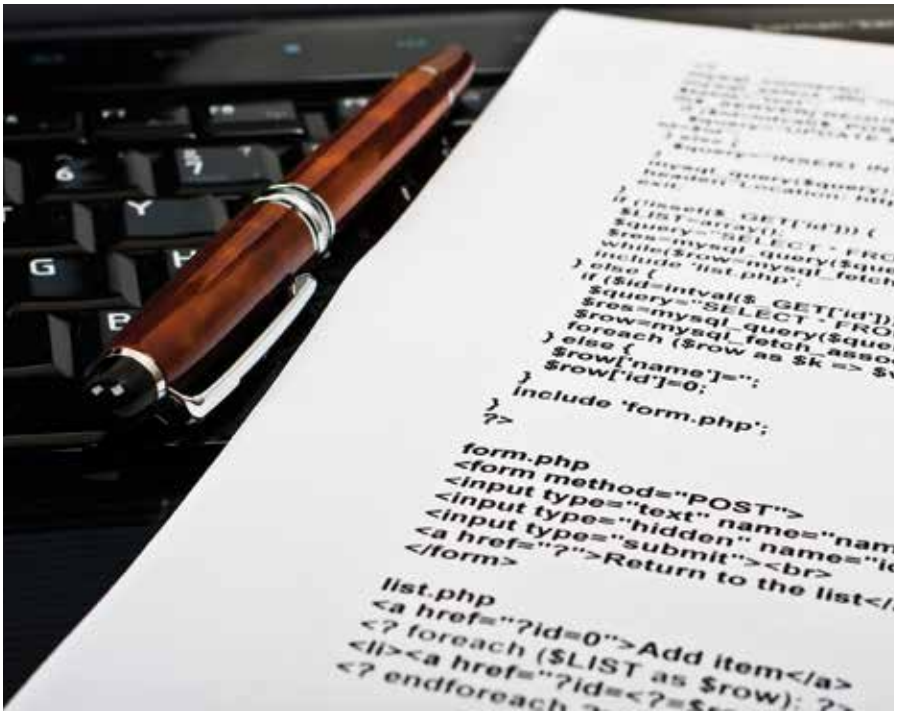
为什么业务分析和大数据对现代企业组织真的很重要。

最近，企业家们无论走到哪里，都能听到有人谈论业务分析或由此衍生出来的如供应链分析、市场营销分析、人力资源分析以及预测、可视化、Web或流数据分析等技术的优点。[5,8]学术界也在倡导关于分析的这一新观点。[6] 以致最近出现一个职位：数据科学家，来实现企业不同部门内和部门间的数据分析和分享。[3]人们甚至担忧，在不久的将来，这类新型专业人员的数量将不足以满足社会对分析专业日益增长的需要。那么这一切真正意味着什么呢?

当今企业充斥着各种数据。[10]在颠覆性技术和组织变革的余波中，企业领导面临着各种强大的力量。例如，随着存储容量和网络速度的提高，信息处理能力变得更强大、更灵活。同时，全球化和其他竞争因素也给企业施加了巨大压力，迫切要求企业提高效率，强化企业和客户之间的关系。在这种竞争中，每个后续的阶段都需要更多的数据和分析，为战略、管理和运营决策提供支持。于是这种竞争迫使企业追求更多更好的分析技术，而分析能力的增强又反过来使得竞争愈加激烈。这种循环导致相互影响的竞争基础与技术进步趋于统一。分析越有效，竞争越激烈；而越激烈的竞争，进一步迫使企业追求更高效的分析能力。技术进步带来更多竞争，更多竞争产生更多技术，更多技术导致更多竞争，如此循环。

真正的变革，还是新瓶装旧酒?

尽管如此，作为一个实际问题，人们仍然容易对业务分析持怀疑态度。[1,7] 这主要是因为所谓的“新”事物是一套广泛使用的完善的分析方式和方法，除了几项改进，毫无新意可言，甚至有些是在历史上某些场合被证实是不可靠或不切实际的。这些新的分析本质上采用了相同的多元推理和描述性统计方法以及数学建模技术，企业很早就开始使用这些技术来分析数据，为复杂的决策提供支持。例如，相关性、聚类分析、筛选、决策树、贝叶斯分析、神经网络分析、回归分



照片由 VITALIY NOVITSKY 提供

真正有意义的是，依赖于分析对于企业现在应采取的竞争方式意味着什么。

析、文本分析等等，都存在于分析库中，并没有特别新的东西。[2] 此外，即使借助今天最现代的技术和工具，这类分析仍有实用限制。例如，软件和数据过于复杂不利于有效的分析，并且复杂分析结果的不同解释甚至可能产生致命的误导。因此，从这个角度人们很难认可业务分析是不同寻常的东西，也难以接受相比众多企业决策者过去惯用的分析方法，最近基于数据的业务分析有什么不同之处。人们甚至认为这种改变实际上只是做了一些增量的变化，并无本质意义的改变。

业务分析的另一个关键方面通常称为大数据。[9] 其特点是大量各种结构化（及非结构化）数据，对其经过深度处理，可以揭示隐藏在数据中的各种现象。[4] 但这同样也是企业高管们一直都在做的事情。还记得“信息超载”这条十年前非常流行的术语吗？处理大数据的基本工具和技术包括数据库管理（尤其是数据仓库）、数据挖掘、仪表板，以及相关技术。在数据管理领域，这些都不是新东西。数据歧义、数据筛选、数据上下文、数据解析、数据转换或数据冗余同样也不是什么新概念。再次，除了数据管理上的一些改进，几乎没有特别新的内容。

理论上，大数据有其不同之处，因为它有三个 V — volume（量）、velocity（速度）和 vari-

ety（多样化）。[10] 也就是说，大数据包含大量数据（量大），数据更新迅速且频繁（速度快），并且在格式和内容上非常多样化（变化多）。这些因素使得企业不得不寻求具有成本效益的创新方法来组织、处理和交付实时信息。有人认为大数据真正显示出与过去的不同。勉强对；但这一观点也在不断变化。数十年来，公司一直在经历数据在量、速度和多样化的不断增长，并且最近随着 Web 的发展呈现出加速增长。除了数据量大以外，这也没什么不寻常之处。尽管如此，累积变化只是这一问题的一个方面。这里的关键点是，这种累积如何在技术之外与企业战略、运营和控制相互作用。

改变的基本驱动力

所以，业务分析为何突然来到了聚光灯下？有一点越来越清楚。这不大可能只是另一个时髦的词，大肆宣传一段时间之后，悄悄退出舞台中心。公司是真的很关注这个问题，并且有真正改变的动力。为什么？

竞争压力与作为核心竞争力的分析和大数据技术之间相互作用，使企业的变革周期越来越长。长期以来，数据收集、操纵、传输和分析技术一直在不断改善。不同的是，这些技术已达到并正在突破数据处理和存储的容量阈值，而组织在处理不断产生的数据方面的传统能力无法突破这个阈值。业务分析技术使组织能够更好地应付这些新的实际处理场景。这非常重要，因为今天在用的基本技术和方法的累积范围及规模反映了对组织产生了一定程度的影响，使大规模分析成为保持企业竞争力和加强日常决策的关键。这里的关键是这些技术和方法的范围与规模，而非分析和数据工具是否为新发明。这场革命是真实的，并且是永久的。

那为什么现在爆发了这场革命？除了使大量数据的累积和处理更具成本效益的技术创新以外，这里另一个关键概念是竞争基础，即企业应不断提高决策能力以在竞争中获胜。[6] 通过对复杂数据进行一致、系统的分析以支持决策，企业能够全方位实现更智能的运营。尤其是近些年来对战略业务分析的强调，不仅提升了执行期望，而且有助于将业务分析目标转化成重要的竞争压力。业务分析方法的应用可提升组织的总体决策能力，从而增强其智能地开展其业务的能力。所以，提升组织智能水平的欲望（以及加速增长的需求）是实施业务分析的主要驱动力。

执行透视

虽然这些观点能够产生很好的共鸣，但对于这种现象可能有更广泛的解释。自从计算（和联网）作为一种职业问世以来，计算专业人士有一个共同的愿景 — 随着时间的推移，这种技术最终完全集成到组织每个部分的每个运营和管理职能。今天的业务分析就是这一愿景的体现。实际上，在现代组织中，计算技术已经成为企业在信息瞬息万变的时代保持基本竞争能力的关键。这一业务分析目标意味着对大规模数据采集和分析的必需，从而能够真正在现代化的全球市场中保持竞争力。毫无疑问，这种依赖也源自计算技术的不断进步；但真正有意义的是，依赖于分析对于企业现在应采取的竞争方式意味着什么。业务分析和大数据不仅仅是营销炒作，或者“旧瓶装新酒”式的统计分析和数据分析方法的延伸，而是未来。就如当今大家所理解的一样，分析是真正意义上的新概念。这一术语就是关于计算将如何永远且不可逆转地挑战和改变商业世界的一个历史愿景的实现，而这一愿景现在穿上了业务分析的外衣。 C

参考资料

1. Barton, D. and Court, D. Making advanced analytics work for you.*Harvard Business Review 90*, 10 (Oct. 2012), 78–83.
2. Davenport, T.H. and Harris, J.G.The prediction lover’s handbook.*MIT Sloan Management Review 50*, 2 (Feb. 2009), 32–35.
3. Davenport, T.H. and Patil, D.J.Data scientist:The sexiest job of the 21st century.*Harvard Business Review 90*, 10 (Oct. 2012), 70–76.
4. Davenport, T.H., Barth, P., and Bean, R. How “big data” is different.*MIT Sloan Management Review 54*, 1, 43–46.
5. Hopkins, M.S., Lavalle, S., and Balboni, F.10 insights:A first look at the new intelligent enterprise survey.*MIT Sloan Management Review 52*, 1 (Jan. 2010), 22–26.
6. Hsinchun Chen, H. Chiang, R.H.L., and Storey, V.C. Business intelligence and analytics:From big data to big impact.*MIS Quarterly 36*, 4 (Apr. 2012), 1165–1188.
7. Jacobs, A. The pathologies of big data.*Commun.ACM 52*, 8 (Aug. 2009), 36–44.
8. Kiron, D. and Shockley, R. Creating business value with analytics.*MIT Sloan Management Review 53*, 1 (Jan. 2011), 57–63.
9. LaValle, S., Lesser, E., Shockley, R., Hopkins, M.S., and Kruschwitz, N. Big data, analytics and the path from insights to value.*MIT Sloan Management Review 52*, 2 (Feb. 2011), 21–22.
10. McAfee, A. and Brynjolfsson, E. Big data:The management revolution.*Harvard Business Review 90*, 10 (Oct. 2012), 78–83.

Charles K. Davis (ckdavis@post.harvard.edu) 是德克萨斯州圣托马斯大学卡梅隆商学院信息管理系的讲席教授。他最近在美国麻省理工学院运输与物流中心担任过访问学者。

译文责任编辑：唐杰

DOI:10.1145/2605176

文章编写主导者 acmqueue
queue.acm.org

“几近函数式”编程并不起作用。

作者：ERIK MEIJER

排中律的诅咒

软件行业开始出现一种趋势，即将“几近函数式”编程视为解决开发者所面临的并发、并行（多核）、以及大数据难题的灵丹妙药。当代命令式语言能够继续保持前进趋势，维持闭包（closure，由函数和引用环境组成的整体），并试图限制变化和其它副作用。可惜，就如同“几近安全”并不起作用一样，“几近函数式”也不起作用。相反，开发者们也应该认真考虑一种原教旨主义选项：接受纯粹的惰性函数式编程，并且使所有副作用显式出现在使用单子（monad，一种数学结构）的类型系统中。

就像节食者们迷恋10分钟神奇锻炼小工具一样，开发者们似乎也已准备好喜欢上这些针对他们所处领域最新危机的简单解决方案。最近，许多人在鼓吹将“近函数式编程”和“有限副作用”作为针对并发和并行等新兴技术的最佳武器。很少人愿意承认这些好处必须以牺牲I/O（输入/输出）等常见操作中潜在副作用的便利性为代价，就像节食者也不愿承认运动的好处必须以时间和汗水为代价。

就像“几近安全”一样，“几近纯粹”只是痴心妄想。命令式编程中细微的副作用就能抹杀纯粹的所有好处，就像一个细菌就能感染消过毒的伤口一样。另一方面，从根本上消除所有副作用（显式和隐式）则会使得编程语言毫无作用。这是排中律的诅咒：你必须认真面对这些副作用，要么接受编程最终是关于变化状态和其它作用，但由于实际原因尽可能抑制这些副作用；要么废弃所有隐式的命令式副作用并使它们显式出现在类型系统中，但由于实际原因偶尔允许显式副作用被抑制。

问题

命令式程序通过重复执行全局共享状态的隐式作用来表述计算。然而，在并行、并发、分布式领域，单一全局状态是不可接受的瓶颈，因此支撑大多数当代编程语言的命令式编程的基本假设开始站不住脚。与通常的看法相反，使状态变量保持不变远未接近消除不可接受的隐式命令作用。类似异常、线程和I/O等普通操作可以像简单可变状态一样导致很多困境。考虑下面从一个数组中过滤出20和30之间值的C#例子（感谢Gavin Bierman），如图1所示。

因为Where有惰性（或使用延迟执行），q0和q1所使用的判定作用是交错的，因此计算q1打印20和30之间的所有值就好像这些判定是交叉的：

1?小于30；1?大于20；25?小于30；25?大于20；[25];40?小于30；5?小于30；5?大于20；23?小于30；23?大于20；[23];

普通程序员肯定会期望q0在q1开始并删除所有小于20的值之前过

滤掉所有大于30的值，因为程序是按照这样的方式写的，两个语句之间的分号证明了这一点。任何判定之间的相互依赖都会出现难以对付的意外后果。

这是一个惰性和异常混杂的例子。如果在传递给Select（映射）函数的闭包体中抛出异常，由于延迟执行作用该异常并不会被抛进try-catch处理程序的范围内。相反，直到foreach循环开始强制计算，异常才会被抛出并且没有相应的处理程序，如图2所示。

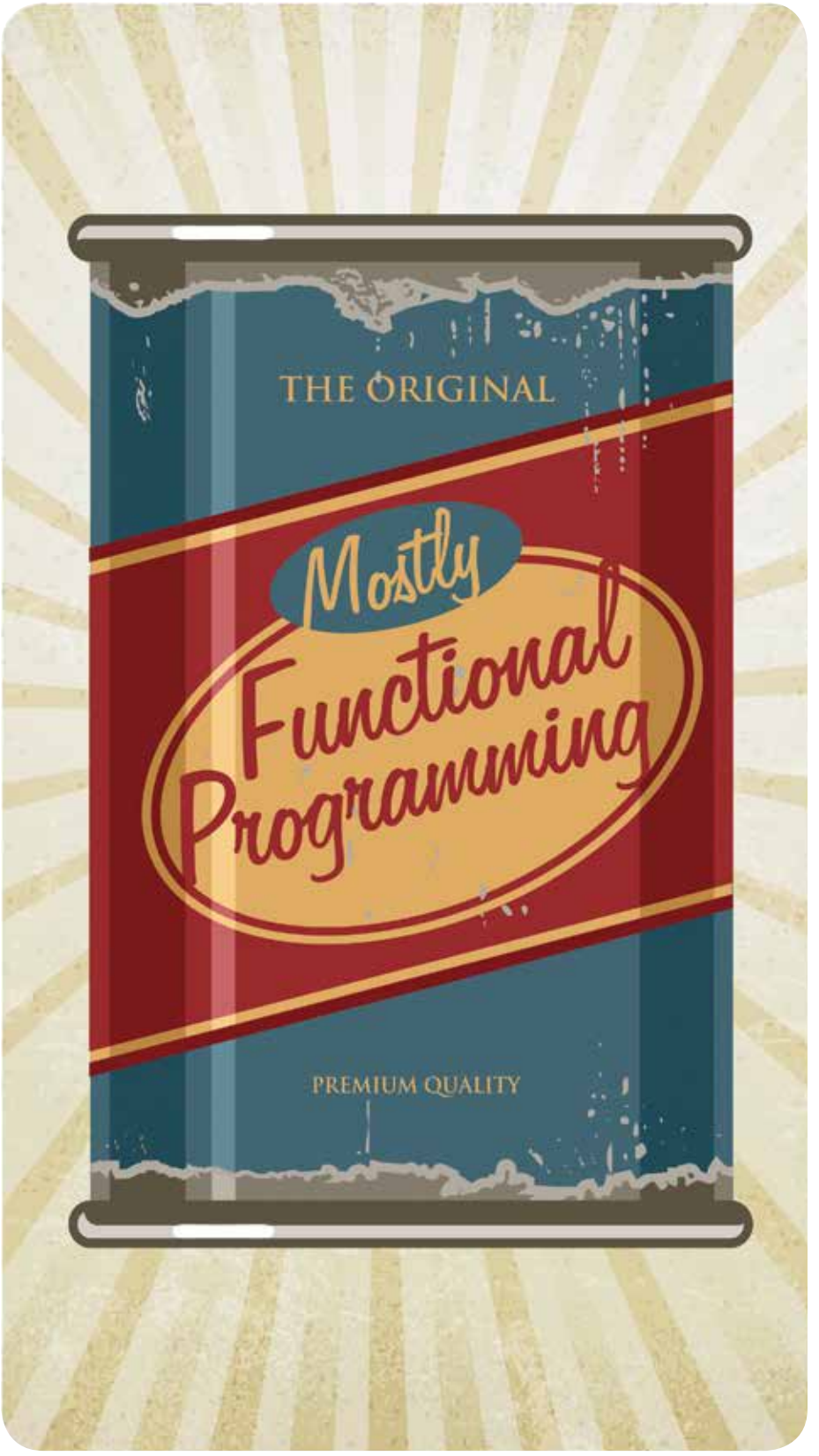
这些作用会在其它方面对语言特征造成干扰，例如闭包和一次性模式之间的相互作用。接下来的例子打开文件进行读取并捕获闭包中超出using块作用范围的文件变量。在C#中，当控制流到达语法块结尾时，using语句释放在语法块入口处自动初始化的变量。因此，等到闭包被调用时，文件已经被处理，从而引发意外异常，该异常在时间和空间上都远离静态源代码中处理异常的代码。

try-catch和using块的作用域并没有与有副作用的闭包的动态范围融合一致。

掉入陷阱

如果带有惰性和闭包的例子看起来太牵强，那么让我们来看一种在printf风格调试中追踪控制台返回值的方法。 这看起来似乎没有问题，但现在对编译器而言执行公共子表达式消除之类的简单操作并不安全。

```
string Ha() { var ha = “Ha”;
Console.Write(ha); return ha; }
```



插图由 ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES 提供

```
// prints HaHa
var haha = Ha()+Ha();

// prints Ha
var ha = Ha();
var haha = ha+ha;
```

但是，实际情况更糟。简单地创建一个新对象都会有可见的副作用。即使用相同的参数调用，构造函数也会每次返回不同的结果，这可以通过GetHashCode或ReferenceEquals方法观察到，其中：

```
var a = new Object();
//例如, 58225482
Console.WriteLine(a.GetHashCode());

var b = new Object();
//例如, 54267293
Console.WriteLine(b.GetHashCode());

Debug.Assert(a != b);
```

为了让构造函数纯粹，你必须为所有对象坚持使用值语义，这意味着在对象中消除所有共享可变状态。这样就失去了面向对象编程的最重要特性：在单个单元中封装状态和行为。

这些例子说明了隐式命令作用的另一种不良后果：它消除了许多常见的优化转换。因为副作用隐性改变了程序全局环境，因此很难分离和定位这些作用。结果，命令式程序大多是非组合的，这使得程序员和编译器都很难解释它们。

这很糟糕，不过消除所有状态变化足以使代码变纯粹。不！遗憾的是，为了控制状态的变化而只是让类中的所有字段都变为只读状态并且不允许给局部变量赋值并不够。这里的Cω程序表明线程能够轻松模拟状态，即使代码中任何地方都没有出现赋值或变量。如图4所示，私有通道Value(T current)利用线程信息队列隐藏的可变状态保存Cell对象的状态，

通过私有通道传递状态被奉为Erlang（一种编程语言）中主动对象模式或观察者模式的基础。图5显示了上述可变Cell程序的Erlang版本。它使用尾递归函数cell(Value)来保存Cell状态。

注意这种Erlang的观察者模式在对对象进行编码时，使用了基于语言中模式匹配、消息发送和递归原语等方式的动态方法分派。你可以利用它实现可变引用，打破了Erlang语言本身并不公开可变状态这一事实。

这些例子仅仅是冰山一角。副作用的根本问题是存在许多不可控的方式来观察它们，更糟的是，通常可以用一种作用来模拟另一种。在纯组合电路中增加递归能够构建触发器来提供可变状态。能够证明状态和分隔延续性足以模拟任何作用，[1]并且未检查的异常可以模拟延续性。[4]在处理这些作用时怎么小心都不为过。

上述例子展示了在最令人意外的地方出现的副作用是多么微妙和棘手。那么，你可能会问“能否通过小心仔细并且遵守规则来避免导致问题的特征，从而将一个命令式语言转化为纯函数式语言？”为了做到这点，你将不得不删除*所有*语言固有的作用。甚至计算也是一种作用，你必须将它的控制权交给编译器和运行时。结果是，这样的超级纯的程序无法与用户或环境交互，不能进行网络I/O、对用户界面事件作出反馈、从文件读取数据、获取当前时间或是产生随机数。这是一个糟糕的两难境地：如果纯的程序不能留下任何曾经被执行的踪迹，那么它们又怎么会有用呢？

虽然情况似乎很严峻，幸运的是有几种摆脱困境的解决方法。他们都涉及精心增加功能而不是盲目删除它们，就像1984年约翰·休斯(John Hughes)在他那篇影响深远的论文中提到的那样，“为什么函数式编程重要。”[2]

原教旨主义函数式编程

纯函数式编程是利用数学函数编程。这意味着表达值之间依赖关系的唯一方式是在参数上应用函数并获取返回值。每次用相同的参数调用函数都会返回相同的结果。这使得很多动作无法进行，包括保守秘密、在一个小地方隐藏一个值并在稍后获取、直接说做那件事前先做这件事、令线程自旋等待、抛出未检查的异常、或在控制台上打印一些东西。这可能看起来比较死板和原教旨主义。确实如此。但这也是有效且可行的。

要理解原教旨主义函数式编程可以如何帮助解决并发问题，重要的是要明白它不仅仅是没有副作用的命令式编程——后者我们已经知道是没用的。相反，它利用数学函数的原教旨主义语言，使用单子表达和封装众多作用组合。为了构建单子的直观模型，我们来看看或许是可能的最简单泛型方法：

```
T Identity<T>(T me) { … }
```

这种方法的类型签名表明*对所有*类型T，给定类型参数T，该方法将会返回类型为T的值。还没完，这种命令式类型签名太*随意*、放任、太松散。它仅仅表达了该方法实际*数学* 类型的一种近似。这并没有包括执行过程中可能隐藏的任何副作用，例如立即求解参数值这一事实，它会检查泛型类型参数（从而并不对所有类型T进行相同的工作），它可能抛出异常，执行I/O，令线程自旋等待，在堆中分配新对象以及获取一个锁，它还能访问this指针，等等。

纯粹原教旨主义泛函语言的秘诀就是有强制类型并将所有作用显式暴露在使用单子的类型签名中，将纯的值类型和可能产生作用的计算类型区分开。

单子的非正式介绍

对于a类型的值，我们来写一个有副作用的计算程序返回一个 a 类型的值，用Haskell的泛型类型语法写作函数应用(M a)的符号中传递这种类型的值。一种设想这种有效计算的方法是使用一台机器产生类型a的输出值，同时显式的显示输出值计算引起的作用M。设想一下真实世界里的工厂在生产商品的同时，产生耗电并污染环境的副作用。注意一个类型为M a的值仅仅是产生类型为a的值的一个承诺，而并没有执行任何作用。为了用单子值做些事情，有两种标准的组合子可以用来进行这种有副作用的计算。

- *中缀应用函数* (ma>>=\a->f(a)) 通常被称为*绑定*，它执行计算ma来产生作用，将其结果称作a并将它传递给函数f。显然，函数应用(f a)的结果又是一个潜在的副作用计算；因此，绑定的类型特征看起来是这样的：

(>>=)::Ma->(a->Mb)->Mb.

- *注入函数* (return a) 向计算中注入纯的值。它的类型签名是：return :: a -> M a。

实际上，每一个作用通常伴随着所谓的*非真态射*。这些特定领域的操作唯一对应这些作用（请参阅文件后面的例子）。

你现在可以将上面表格中的所有有副作用计算抽象并形式化为一种被称为*单子*的代数结构。它支持创建和传播效应的两种通用操作：注入和绑定 (>>=)：

class Monad m where {

图 1.混合延迟计算选择

```
static bool LessThanThirty(int x) {
  Console.Write("{0}?Less than 30;", x); return x < 30;
}
static bool MoreThanTwenty(int x) {
  Console.Write("{0}?More than 20;", x); return x > 20;
}

var q0 = new[]{ 1, 25, 40, 5, 23 }.Where(LessThanThirty);
var q1 = q1.Where(MoreThanTwenty);
foreach (var r in q1){ Console.WriteLine("[{0}];",r); }
```

图 2.尝试通过零异常获取分隔。

```
var xs = new[]{9,8,7,6,5,4,3,2,1,0};
IEnumerable<int> q;
try { q = xs.Select(x=>1/x); } catch { q = new int[]; }
foreach(var z in q){ Console.WriteLine(z): // throws here }
```

图 3.使用已关闭的文件。

```
Func<string> GetContents;
using(var file = FileSystem.OpenTextFileReader(@"my file")) {
   GetContents = ()=>file.ReadToEnd();
}
Console.WriteLine(GetContents()); // 意外！出现异常
```

图 4.连接+线程=可变状态。

```
class Cell<T> {
   Cell<T>(T init){ Value(init); }
   T Get() & async Value(T current){ return current; }
   async Set(T @new) & async Value(T old){ Value(@new); }
}
```

你可以使用类型Cell的readonly域定义可变Point类：

```
class Point {
   readonly Cell<Int> x = new Cell<int>(0);
   readonly Cell<int> y = new Cell<int>(0);
}
```

图 5.尾递归和线程=可变状态。

```
new_cell(X) -> spawn(fun() -> cell(X) end).
cell(Value) ->
   receive
      {set, NewValue} -> cell(NewValue);
      {get, Pid}      -> Pid!{return, Value}, cell(Value);
      {dispose}       -> {}
   end.
set_cell(Cell, NewValue) -> Cell!{set, NewValue}.
get_cell(Cell) ->
   Cell!{get, self()},
   receive
      {return, Value} -> Value
   end.
dispose_cell(Cell) -> Cell!{dispose}.
```

```
    (>>=) ::ma->(a->mb)->mb
    return::a                    ->ma
}
```

Haskell所有单子的母类是I/O单子，它代表所有拥有全局作用的计算。因此，它伴随着大量的非真态射，对命令式编程程序员而言其中大部分都很熟悉，例如向控制台进行读写操作，令线程自旋等待以及抛出异常。一旦作用必须在I/O单子中显式表达，如分配变量以及相关的读写操作就必须被提升进入I/O单子（如图6）。

从forkIO类型中你能够直接看出令线程自旋等待是一种有副作用的操作。它确保了任何使用线程的可变单元编码也都处于I/O单子中。这避免了程序员误认为他们在定义不可变类型。注意I/O单子并没有阻止全局状态同时被多个线程更新，如forkIO类型所示。

单子的真正威力来自计算自身也是值这一事实，它们能在纯的宿主语言中作为一等公民被传递。这允许程序员使用单子原语和非真态射写出新的抽象（特定领域和自定义控制结构）。

例如，你能够轻易定义带有一系列副作用计算的函数，执行其中每一个计算并将结果收集为一个列表，如图7所示。

纯粹的原教旨主义函数式语言对内部DSL（Domain Specific Languages，领域特定语言）而言是一种非常方便的宿主语言，因为它仅是可执行的指称语义。因此，向着实现P.J.Landin在1966年发表的有影响力的论文中“接下来的700种编程语言”的愿景，纯泛函语言前进了一大步。

让作用显式出现在类型系统中还有一个额外好处，那就是你可以分辨不同的作用并自由引入新作用。例如，你可以定义事务型内存单子(STM a)。它包含分配、读取和写入事务型变量的非真态射，如图8所示。

既然从I/O到STM没有（隐式）泄露，STM单子可以捕获执行事务所需的机制而不用操心对任意作用的回滚，因为可以假设普通的Haskell计算是纯的。

副作用不仅难于控制，而且经常偷偷出现。即便在据信为纯的Haskell中，也有一个名为unsafePerformIO :: IO a->a看似不起眼的函数。它告诉编译器在计算参数时“忘记”所包含的副作用。这个“几近纯的函数”应该用以在一个其他部分都是纯的计算中封装善意的副作用；然而这个unsafePerformIO却打开了一个潘多拉盒子，因为它颠覆了Haskell的类型系统，破坏了语言保证并允许任何类型转换为其他类型中。

```
unsafeCast::a->b
unsafeCastx=unsafePerformIO(do{
    writeIORef castref x; readIORef
    castref
})
castref = unsafePerformIO (do{
newIORefundefined})
```

这些例子表明了使用单子把有副作用的计算当做值的威力，但普通开发者能否处理这些的问题仍然是一个问题。我们认为答案完全是肯定的。LINQ（语言集成查询）成功恰恰源于单子的事实也证实了这一点。LINQ标准查询操作本质上是单子操作。例如，SelectMany

图 6.作用禁区。

```
数据IO a –不需要知道这是如何实现的。

putChar ::Char -> IO ()
getChar ::IO Char

newIORef   :: a             -> IO (IORef a)
readIORef  ::IORef a       -> IO a
writeIORef ::IORef a -> a -> IO ()

forkIO ::IO a -> IO线程
```

图 7. Haskell，世界上最好的命令式语言

```
sequence           ::Monad m => [m a] -> m [a]
sequence []        = return []
sequence (ma:mas) = ma >>= (\a -> sequence mas >>= (\as -> a:as))
```

图 8. 孤立事务

```
newTVar   :: a            -> STM (TVar a)
readTVar  ::TVar a       -> STM a
writeTVar ::TVar a -> a -> STM ()

此重试操作中止当前事务并等到任何事务性变量被改变，或当通过orElse运算符链接时它会尝试运
行另一个事务块。

retry  ::STM a
orElse ::STM a -> STM a -> STM a

最终，函数atomic将事务单子注入普通I/O单子中。在这一点上，值上无法执行更多事务操作。

atomic ::STM a -> IO a
```

扩展方法直接对应之前介绍的Haskell绑定(>>=):

```
(>>=)::ma->(a->mb)->mb
M<T> SelectMany(this M<S> src,
Func<S,M<T>>f)
```

Haskell的多态和公共语言运行库(CLR, Common Language Runtime)、Java或其它面向对象语言中泛型的主要不同在于，Haskell中单子的定义依赖更高阶的多态，即单子类（接口）的参数化使用类型构造函数而不是简单的类型。通常，泛型仅允许通过类型参数化——如List<T>和Array<T>，但不允许通过形如M<T>的容器类型M参数化并将M实例化为List或Array。因此，编码单子的LINQ标准序列操作必须依赖句法模式，根本上的过载，而不是适当的泛型。

使您可以在作用上定义可重用抽象的好处是为什么人们通常叫Haskell是世界上最好的命令式语言。或许工业界面临的真正危机比采用纯原教旨主义函数式编程的疼痛要糟糕得多。对LINQ的热捧证明人们已经准备好做出改变。

替代方案

原教旨主义方式编程的共同信念是普通程序员的范式转变太多，前进的道路是通过抑制作用使现有命令式语言更纯。推荐的办法是从两方面来控制作用：假定所有方法都有对环境的副作用（即在I/O单子里），除了那些被特殊修饰语标记的方法，例如threadsafe、pure、immutable、readonly、区分val和var等以及表明缺乏事件等等。对开发者而言，乍一看这可能没有那么碍眼，但可以说情况并非如此，因为一旦处于纯的上下文中，你无法调用非纯函数，因此也必须标注在纯方法中被直接或间接调用的任何方法，就像一个Haskell纯的表达式深处增加作用需要使用单子风格完全重构代码一样。

纯函数标注的一个问题是它们不可扩展，即用户不能定义新的“无作用”。我们已经在单子组合的例子中看到，在需要的时候支持用户定义其作用和非真态射很有必要。

第二，纯函数标注通常依附于*函数*，而在Haskell中，作用并没有与函数绑定，反而与*值*绑定在一起。在Haskell中，类型f::A->IO B的函数是纯函数，它在给定A类型的值时会返回以IO B类型的值表达的副作用计算。然而，应用函数f不会造成任何直接的作用。这与标记纯函数大不相同。如图所示，在值中附加作用允许程序员定义他们自己的控制结构，例如，将多组副作用计算的列表置入计算列表的副作用计算中。

有许多其它建议来探讨命令式语言中的作用，例如线性或唯一类型、所有者类型或最近的分离逻辑。[5]然而，它们都对用户和工具的复杂性提出了较高的要求。这是十分复杂的数学方法，特别是与简单的单子等式推理和纯函数式编程相比，因此它并没有简化普通程序员的工作。编写和理解代码并不应该需要理论计算机科学博士学位。

对开发者而言，抑制作用使命令式语言变纯与成为原教旨主义者并使纯语言成为命令式一样痛苦。

结论

“几近函数式编程”的想法并不可行。仅仅通过部分删除隐式副作用并不能使命令式编程语言更安全。遗留一种作用通常足以模拟曾经试图删除的那种作用。另一方面，允许在纯的语言中“忘记”作用也容易以自己的方式引起混乱。

不幸的是，没有恰当的平衡，并且我们面临着一个典型的二元论：排中律的诅咒。它表示选择尝试使用纯函数标注控制作用，并接受代码本质上仍然会产生效应的这一事实，或让所有作用显式出现在类型系统中从而接受纯粹并通过引入非函数例如unsafePerformIO从而更加务实。这些例子是为了说服语言设计者和开发者扩大视野并开始更认真看待原教旨主义函数式编程。 C

queue.acm.org 上的相关文章

与Erik Meijer和Jose Blakeley的谈话
http://queue.acm.org/detail.cfm?id=1394137

Hop多层编程
Manuel Serrano and Gérard Berry
http://queue.acm.org/detail.cfm?id=2330089

Masses FPGA编程
David Bacon, Rodric Rabbah and Sunil Shukla
http://queue.acm.org/detail.cfm?id=2443836

参考资料

1. Filinski, A. Representing monads.In *Proceedings of the 21st ACM Symp. on Principles of Programming Languages* (1994).ACM Press, 446–457.
2. Hughes, J. Why functional programming matters. *Computer J.32*, 2 (1989), 98–107.
3. Landin, P.J.The next 700 programming languages. *Commun.ACM 9*, 3 (Mar. 1966), 157–166.
4. Lillibridge, M. Unchecked exceptions can be strictly more powerful than call/cc.*Higher-Order and Symbolic Computation 12*, 1 (1999), 75–104.
5. O’Hearn, P.W.A primer on separation logic (and automatic program verification and analysis).*Software Safety and Security; Tools for Analysis and Verification. NATO Science for Peace and Security Series* 33 (2012), 286–318.
6. Oram, A. and Wilson, G. *Beautiful Code:Leading Programmers Explain How They Think*.O’Reilly Media, 2007.

Erik Meijer (emeijer@applied-duality.com)是Applied Duality公司创始人和代尔夫特工业大学(TU Delft)云计算教授。他以对Haskell、C#、Visual Basic和Hack等编程语言的贡献以及在LINQ和Rx Framework的工作而闻名。

译文校对：肖天

译文责任编辑：陈文光

DOI:10.1145/2602575

EPIC 助力评估网络威胁在网络及物理维度上对网络化关键基础设施的影响

作者：CHRISTOS SIATERLIS，BÉLA GENGE

网络-物理测试床

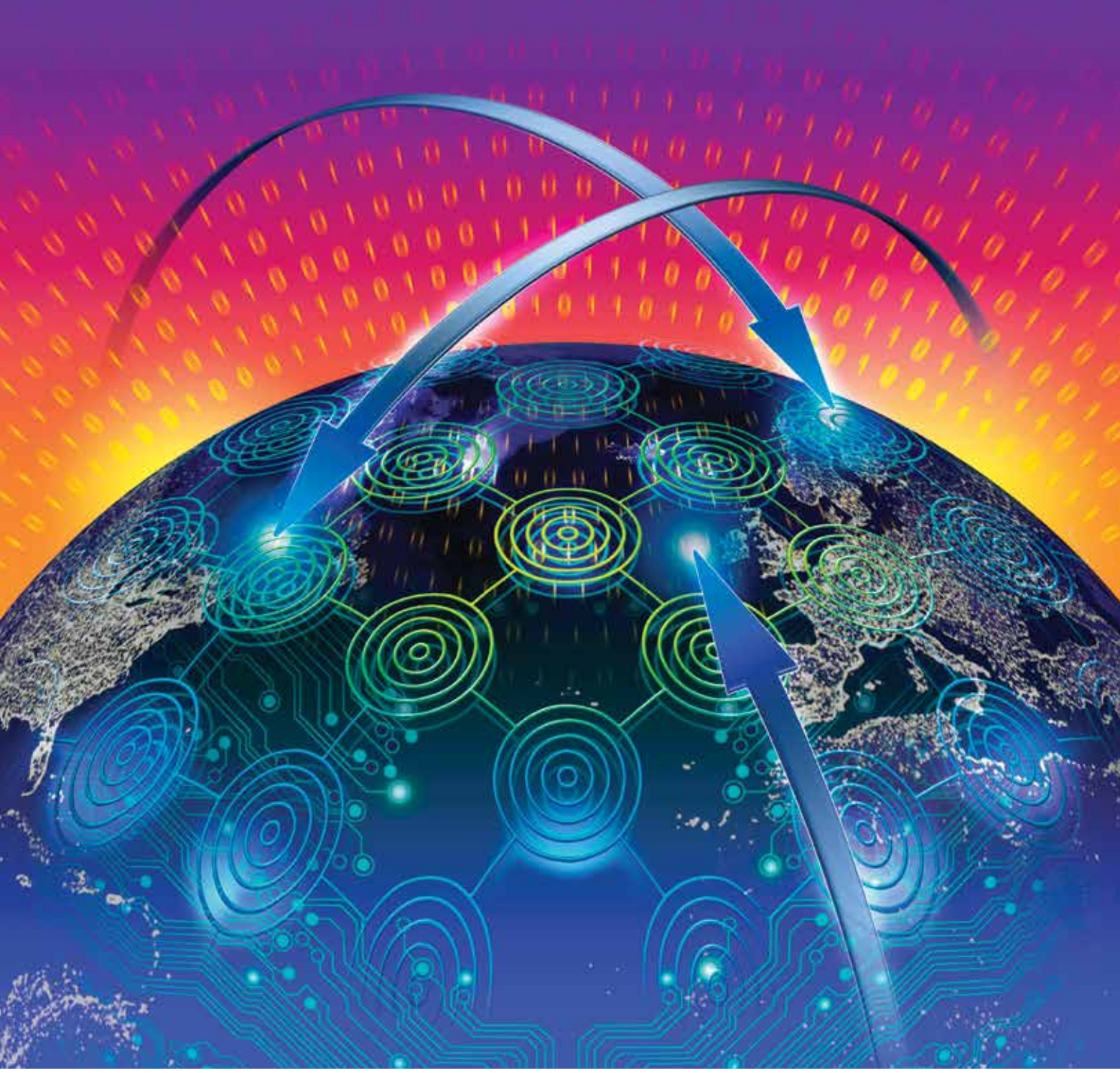
现代社会的顺利运行依赖于网络化关键基础设施（NCI）所提供的服务的质量和可靠性。物理基础设施，包括交通系统、电网和电信网络，为社会提供了基本的服务，保障了经济的平稳运行，满足了公民的生活所需。不过，偶然发生或者蓄意造成的故障却成了现代生活中最严峻风险的之一。

在过去几年中，网络化关键基础设施中使用的信息和通信技术（ICT）激增。各公司采取上述措施的主要原因是为了降低工业设施的成本和推出新服务（比如对基础设施进行远程监视和维护，能源市场，以及新兴的智能电网）。尽管优势明显，缺点也不容忽视。广泛使用标准的技术组件后，NCI会面临严峻且常见的网络威胁；例如，利用恶意软件发起的蓄意攻击，[6] 或是因配置错误和软件错误而导致的[5]无意威胁，均可能引发严重的服务中断。在有关数据采集与监视控制系统（SCADA）安全的多项研究和报告中，研究人员已经强调了上述问题。数据采集与监视控制系统[6,15] 代表了核心的NCI基础设施，监视和控制了多个物理过程。该系统主要由执行机构、传感器以及执行物理动作的硬件设备（如打开阀门）组成，其中还包括监视和控制物理过程的ICT设备和软件。在传统的ICT系统中，破坏性的网络攻击的影响往往限制在网络操作的范围之内。与此不同，在设有关键基础设施资产的场景中，此类攻击可能会造成重要服务（例如交通、供水和供气）的中断。如果要从NCI的物理和网络维度评估网络威胁的影响，那么研究人员需要一套精确的科学工具来进行实验验证和测量。

网络-物理测试床积极支持科学方法，是此类工具的一个例子。在测试床的开发过程中，可能会使用

» 重要见解

- **EPIC 允许研究人员开展科学的、严格的网络-物理安全实验，能保证实验的逼真度、可重复性、测量精度和安全开展。**
- **通过利用真实网络、协议以及在基础设施物理维度方面的网络和软件模拟器中包含的软件，EPIC重建了关键基础设施各组件之间的相互依赖关系。**
- **EPIC代表了一类新的科学工具 - 网络-物理测试床 -适于在包含各种基础设施的多种场景下测试网络攻击、验证新的反制措施、以及培训用户。这些场景涵盖了从本地物理过程（比如电厂）到全国性的电网和输电系统的各种场景。**



插图由 ROY WIEMANN 提供

真实系统、仿真器或软件模拟器。不幸的是，如果在生产系统中进行安全性和恢复能力的实验，可能会对关键业务服务造成不良影响。[10]与此类似，如果采用真实的组件搭建专用于实验的基础设施，也存在安全风险[10]，且搭建成本相当高。[11]基于软件的模拟提供了一条有效的途径来研究物理系统，可提供成本低廉、速度快且准确的分析结果。然而，考虑到计算机网络的多样性和复杂性，软件模拟在网络安全方面的应用有限。软件模拟器可以有效地模拟正常的网络情况，但却无法捕获计算器网络的故障发生过程。[7]不过，从另一方面来说，在很多情况下，仿真器不仅可以判断系统是否会发生故障，还能捕获系统的故障发生过程。

为了满足对网络-物理测试床的要求，我们提出了一套名为互联网突发事件验证平台（EPIC）的科学工具。我们开发的这套工具用于从网络和物理维度评估网络攻击对NCI的影响，结果准确，可重复。为了构建NCI的复杂度模型，EPIC使用了基于Emulab（Emulab是用于构建仿真测试床的高级软件套件）的计算机测试床[20,24]以重建NCI中的网络元素以及物理组件的SSim。（术语Emulab 指盐湖城犹他大学的一处设施，也指一种软件）。

动机

现有测试床的主要局限是无法在多个异质的NCI上运行安全性实验；

表1测试床的特性与成本效益的对比：对于某个特定的特征，••• =支持程度高；•• =中等支持程度；以及• = 支持程度低

	逼真度		可重复性		测量精度		安全性			
	网络	物理	网络	物理	网络	物理	网络	物理	成本效益	多个关键基础设施
真实网络设施，真实物理设施										
NSTB[23]	•••	•••	••	•	••	••	•••	•	•	•
ENEL-JRC[15]	•••	•••	••	•	••	••	•••	•	•	•
真实网络设施，模拟物理设施										
EPIC	•••	••	•••	•••	•••	•••	•••	•••	••	•••
DEFT[25] (DETER+VPST)	•••	••	•••	•••	•••	•••	•••	•••	•	•
PowerCyber[11]	•••	••	••	•••	••	•••	•••	•••	••	•
真实与模拟网络设施，真实物理设施										
TUB[8]	••	•••	••	•	••	••	•••	•	•	•
真实与模拟网络设施，模拟物理设施										
VCSE[14]	••	••	••	•••	••	•••	•••	•••	••	•••
模拟的网络设施，模拟的物理设施										
SCADASim[17]	•	••	•••	•••	•••	•••	•••	•••	•••	•••
HLA[16]	•	••	•••	•••	•••	•••	•••	•••	•••	•••

表2利用EPIC开展的典型实验

目标	节点	攻击类型	物理过程	结果
评估网络参数（延时、丢包率及网段划分）对针对NCI的网络攻击的影响	6–15	欺骗	发电厂和化工厂	实验证明，通信延时和丢包率对网络攻击的影响很小，但是，与物理过程有关的网段划分却能提升NCI的弹性。据此，我们开发并验证了一个新颖的网段划分方法。
验证信息物理系统异常监测的新方法	21	分布式拒绝服务（DDoS）和欺骗	电网	在概念验证原型中，研究人员展示了融合后的网络物理异常监测的性能远远优于传统方法，因为其隔离了物理因素与网络因素。本实验促成了对该概念验证原型的验证。
评估操作决定的影响及其对相互依赖的NCI的级联影响	12	协同攻击	电网和铁路运输	它们说明了网络干扰（如失去监测和控制能力）从一个NCI向其他依赖于它的NCI的传播过程，还阐明了NCI操作员必须开展协作，确保互联的NIC的整体稳定性。
分析在广为人知的（YouTube）BGP路由器劫持事件中操作员的反应以及协作的效果	32	劫持和中间人		它们强调了下列因素的重要性：快速侦测BGP劫持事件的工具及相关机制；掌握安全专长且训练有素的操作员；支持在网络服务提供商之间进行通信和协作的可信媒介。

NCI的互联和相互依赖程度相当高，也就是说，某个NCI出现单一故障后，可能会对其他的NCI造成级联影响；例如，在2012年7月，印度北部的电网瘫痪后，影响了6亿多人，造成了交通、医疗保健以及众多其他服务的电力中断。研究人员必须在实验室环境中重建、分析和了解此类场景（此类场景也有可能由网络攻击引发），以便摸索出必要的安全措施应用于真实世界的各项设置。

在NCI的网络元素和物理元素之间存在着多种关键联系。通过重建这些联系，EPIC可应用于多种性质不同的研究。除了开展不同技术的漏洞测试、影响分析和验证之外，EPIC还提供了各种工具来对网络-物理实验中的重要一环（人工操作员）进行闭环。在包含NCI的场景中，人工操作员可帮助系统确保物理过程的稳定性及其正常运行。作为实验的一部分，人工操作员可以直接与EPIC互动，也可以通过构建标准操作规程的模型来模拟他们的操作。无论采取哪种方式，EPIC均可用于构建复杂的实验，测试人工操作员执行的指令对

物理过程的影响，或者度量在某个物理过程的状态发生改变时，人工操作员的反应情况。通过更接近真实NCI操作的精确实验，EPIC是实验测试床领域的新进展。

对测试床的要求网络-物理测试床必须符合科学方法的要求，并能积极地支持科学方法的应用。通过保证实验的逼真度、可重复性、测量精度和安全开展，它还能让研究人员应用严格的科学方法：[20]

*逼真度*实验测试床必须能够尽可能准确地重现拟研究的真实系统。不过，在很多情况下，完全准确地重现真实系统的所有细节或许没有必要，甚至可能无法实现。因此，研究人员更加喜欢真实度可调的实验平台，这意味着在测试实验假说时，研究人员使用的细节程度可以做到恰如其分；例如，在某个实验中，可能需要使用真实的路由器再现物理层的网络，而在另一个实验中，软件路由器或许足以达到要求。真实度可调的概念意味着研究人员可以进行选择，在真正需要真实硬件的时候，使用真实的硬件；在不需要的时候，则可使用仿真器、模拟器或其他的抽象工具。

*可重复性*这种要求反映了在研究人员重复实验时，可取得相同的结果或一致的统计结果这一需要。可重复的实验需要一个受控环境。但是，为了实现这些目标，研究人员必须首先确定实验的初始状态和最终状态，以及中间发生的所有事件。为了重现之前保存的实验场景，研究人员必须能够设置实验平台的初始状态，然后按正确的顺序和发生时间触发所有必要的事件。

*测量精度*实验应该是可以被监视的，但在监视时，不能采用可能干扰实验，导致实验结果发生改变的方式。因此，研究人员需要隔离控制、测量和实验过程。

*安全开展*在很多情况下，安全性实验假定存在着利用恶意软件实现目标的各种对手。软件的影响往往无法预料，有可能会破坏物理系统。但是，在实验中必须重建这些实例，却又不能危害物理测试床本身，或是伤害研究人员。

在选择模拟步骤之前，EPIC研究人员必须验证实时模拟的输出是否尽可能准确地再现了对应的真实世界过程的输出。

现有方法为了评估目前的技术水平，我们回顾了相关文献，然后依照之前确定的需求集（见表1）评估了可获得的各种测试床的特征。

美国管理的美国国家SCADA测试床项目美国能源部[23]实施了国家联合实验室项目，旨在支持业界和政府部门的工作，提高工业设施的网络安全。该实验室提供了一系列的设施来重建真实世界系统，范围包括从发电站到输电网络的各类设施，其中包括真实的电网组件以及业界专用的软件产品。

虽然美国国家SCADA测试床（NSTB）有助于确定漏洞，加固控制系统的防护机制，但由于部署类似设施的成本高昂，所以限制了它在跨领域异构信息物理系统中的实际应用。

意大利国家电力公司（Enel S.p.A.）是意大利最大的电力公司，位于意大利的欧盟联合研究中心（Joint Research Centre）则是欧盟执行机构下属的科学和技术部门。这两大组织携手于2009年开发了一个受保护环境来重建真实燃气轮机发电厂的物理特性。[15] 该测试床准确地重现了普通发电厂的网络和物理特性，包括缩小的物理过程、典型的现场网络、过程网络、安全区、通用型服务、企业域和标准软件。该测试床用于在安全的环境中分析攻击场景，并测试反制措施。不幸的是，由于灵活性差，维护类似架构的成本高，抵消了纯物理测试环境具有的高逼真度优势。

基于Emulab的网络防御技术实验研究（DETER）测试床[2] 提供了可重复的，与安全相关的实验，其中部分为通过DETER实现的联合测试床（DEFT），用于互联在网络-物理

实验中位于不同地域的测试床。在DEFT的联盟中，2009年DETER与伊利诺伊大学开发的虚拟电力系统测试床（VPST）进行了互联[25]。[3]VPST具有通过实时模拟器（比如PowerWorld，一种电力系统专用的模拟器）模拟电网的能力。其对DETER的能力进行了拓展，以开展信息物理系统实验。

EPIC和DEFT的核心区别为，EPIC提供了一个可扩展的、成本效益高的解决方案，可用于开展跨领域异构的物理过程的实验（通过其软件模拟器），而DEFT更专注于特定的基础设施（比如电网）。EPIC可以看作是DEFT计划的补充，因为为EPIC开发的软件模拟器可以很容易地通过DETER重用。

为了模拟电网，爱荷华州立大学[11]2013年开发了PowerCyber 测试床，对数据采集与监视控制系统（SCADA）的专用硬件及软件和实时数字模拟器进行了整合。它使用了虚拟化技术来处理可扩展性和成本问题，爱荷华州立大学还开发了互联网尺度的事件和攻击生成环境（Internet-Scale Event and Attack Generation Environment）来进行广域网的仿真。该测试床还提供了非实时的模拟能力，主要用于模拟较大型的系统及开展状态估计和事故分析。

北京的清华大学采用了另一种方法。他们采用真实组件模拟物理维度，采用部分模拟的组件模拟网络维度，[8] 他们使用了真实的SCADA控制服务器、NS-2 网络模拟器以及真实的控制硬件和现场设备。该测试床设计用于确定网络攻击对SCADA系统的影响，网络攻击包括数据包伪造、被破坏的访问控制机制以及被破坏的SCADA服务器。虽然该测试床提供了可靠的实验数据，但是，由于其中几乎全部的东西都是真实的，所以它无法支持针对大型基础设施的测试（比如全国范围的电网）。

NCI的互联和相互依赖程度相当高，也就是说，某个NCI出现单一故障后，可能会对其他的NCI造成级联影响。

桑迪亚国家实验室开发了虚拟控制系统环境（VCSE）测试床[14]以检测漏洞、训练操作员和验证迁移技术。VCSE采用了名为OPNET的计算机网络性能分析软件以集成真实设备与被模拟网络和PowerWorld（PowerWorld作为该测试床的电力系统模拟器）。VCSE 还包含了Umbra。Umbra是桑迪亚的专利框架，它提供了一个集中的环境来监视和控制多个被模拟的组件。

澳大利亚墨尔本市墨尔本皇家理工大学开发了SCADASim框架[17]。该框架提供了各种预先定义的模块用于构建SCADA模拟环境，其使用了OMNET++离散事件模拟引擎重建典型的SCADA 组件，同时还提供了底层的跨模型通讯层。SCADASim支持通过实现业界标准协议的模块与真实设备进行集成。它可用于开发一系列的SCADA模拟环境和评估网络攻击对通讯以及物理过程的正常运作的影响。

最后， 苏黎世的瑞士联邦理工学院利用了系统的系统（system-of-systems）方法开发测试床，[16]其中使用了高层体系架构（High Level Architecture）模拟标准以提供跨领域实验环境，进行跨领域模拟器的互联。在关键基础设施之间存在复杂相互依赖的背景下，此测试床支持开展假设情景分析。不幸的是，该方法也许在研究互相依赖时有效，但却无法准确地重建网络层。

EPIC概述

EPIC的架构中包含了基于Emulab软件的仿真测试床[20,24]以重建NIC的网络维度，并用软件模拟物理维度。采用基于仿真的测试床后，EPIC确保了逼真度、可重复性、测量精度以及网络层的安全。在攻击时或发生故障时，ICT组件的反应较为复杂。EPIC选择了网络安全领域[2]中广为接受的方法，以克服在

模拟ICT组件行为时面临的主要困难。在物理层方面，EPIC也用了模拟，因为模拟方法有效、安全、成本低廉，且能提供快速准确的分析能力。虽然模拟会放松对逼真度的要求，但是软件模拟能让研究员针对多个异构物理过程开展破坏性实验。不仅如此，我们在文献中发现了多个物理系统的复杂模型。把这些真实的物理系统集成进软件模拟器后，可以准确地再现这些真实系统的行为；比如，由于能源领域内模拟的结果相当准确，备受信赖，所以通常采用模拟的方法来帮助输电系统的运营商做出决策。

重建信息系统Emulab[24]是用于仿真测试床的高级软件套件。在全球范围内，Emulab的私有安装包数量庞大，并赢得了多所大学的支持。2009年，我们使用Emulab 架构和软件开发了第一个安装包。现在，通过Emulab 架构和软件，我们对它进行了持续改进和扩展（见图1a）。EPIC采用Emulab后，我们可自动和动态地把物理组件（比如服务器和交换机）映射到虚拟拓扑中；也就是说，Emulab 软件可配置物理拓扑来尽可能透明地仿真虚拟拓扑。Emulab 的基本架构包括两台控制服务器：使用了一个物理资源池作为实验节点（比如通用PC和路由器），并用一组交换机把这些节点连接起来。Emulab 软件提供了Web 界面来描述EPIC 测试床中实验的生命周期内所包含的各个步骤。

*虚拟网络拓扑*EPIC 研究人员必须先创建对虚拟网络拓扑的详细描述，即“实验脚本”；使用形式化语言进行实验设置后，如果其他研究人员希望重现结果，那么他们重建类似配置的过程会更容易一些。

*Emulab 软件*实验通过Emulab软件进行初始化，自动从可选的组件中预留和分配所需的物理资源。

*实验节点*该软件可通过配置网络交换机连接位于多个虚拟局域网的实验结点来重建虚拟拓扑，然后再为预先确定的链路配置数据包捕获机制来实现各种监视目标；以及

定义的事件。专用的实验软件（比如模拟器）可通过在实验脚本中定义的事件自动启动，或通过登录各节点手动启动。

重建物理系统图1b勾勒了EPIC中用于重建物理系统的软件单元。物理过程模型采用Matlab

图 1EPIC测试床的架构：（a）概述和实验步骤；及（b）软件模块，包括SSim和代理组件。

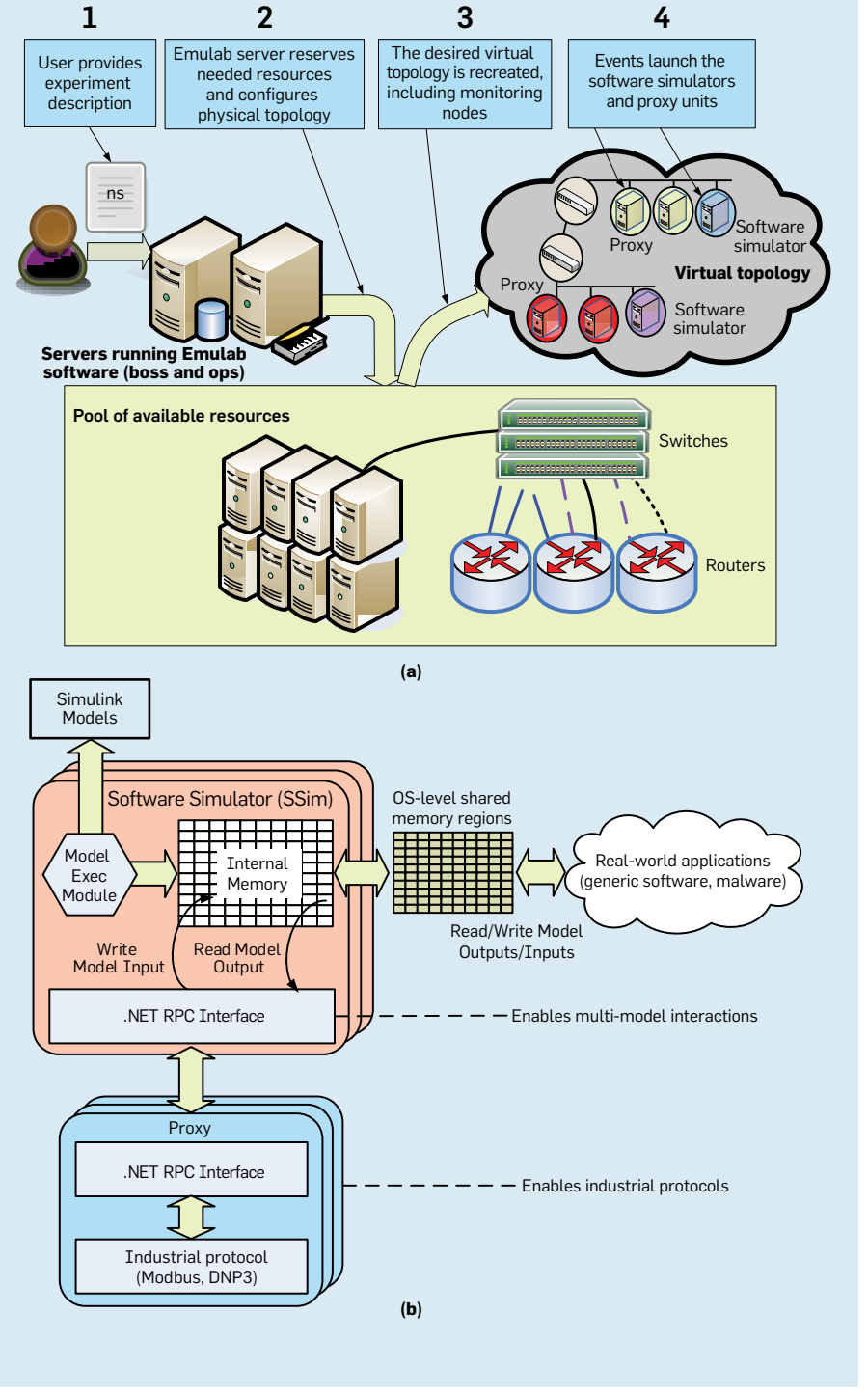
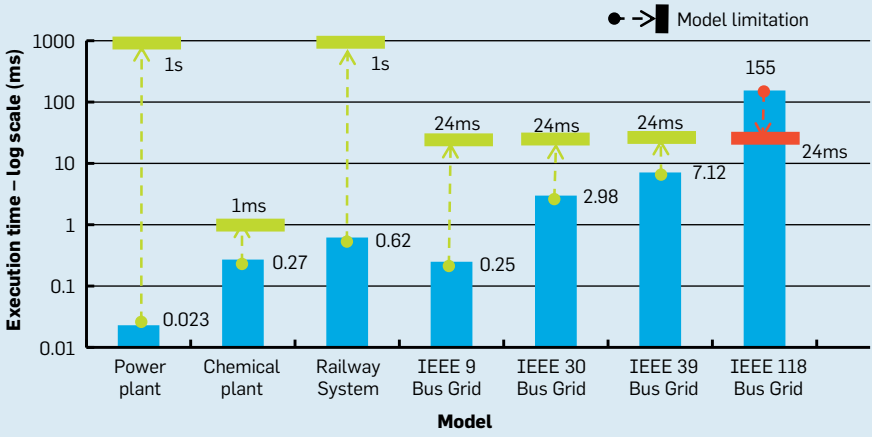


图 2各模型在2.8GHz CPU上的执行时间及其局限；EPIC支持研究人员开展电厂、化工厂、铁路系统的实验以及从标准IEEE模型套件中选取电网模型进行实验。

IEEE 118客车模型中的红线强调，因为模型的执行时间超过了模拟步骤的最大值（或是在模型必须至少运行一次的情况下，模型动力学允许的模型特有的时间间隔或粒度），所以无法满足实时模拟的需求。



Simulink搭建，其对应的C代码由Simulink编码器（Simulink Coder）生成。然后，把生成的代码集成进入软件模拟单元（SSim）中，便可让模拟的过程与仿真测试床的其他部分进行实时互动。从核心层面来说，SSim单元绑定了网络和物理层。从SSim的角度来看，模型等同于黑箱，其输入和输出被动态地映射到内部的内存区域。写入该区域的值被复制到模型的输入中，而模型的输出则被复制回内部内存中。采用这种方式后，EPIC 允许研究人员利用各种物理过程开展实验，而不需要提供这些过程的细节。为了让研究人员研究多个NCI之间的相互依赖，SSim实现了远程过程调用（RPC）接口供其他SSim 实例访问。RPC提供了对内部内存区域的访问，所以能访问模型的输入和输出，进而实现模型间的实时互动。不仅如此，EPIC 还可通过代理组件支持工业协议（比如Modbus）。这些代理组件对SSim 与其他单元（包括工业设施中的服务器）间的调用进行转换。

集成真实世界中的硬件和软件。由于几乎所有的组件都是真实的，EPIC支持通常可在普通PC上运行的任何软件，而且基本上可以集成配有以太网接口的任何硬件；例如，基于EPIC的测试床可包含用于研究特定工业架构的真实的控制硬件和真实的工业软件。真实世界中的软件与EPIC软件单元之间的互动采用下列几种方式实现：第一，真实的软件通过工业协议（比如Modbus）与模拟的模型互动。Modbus 调用被发送给一个代理组件，然后由代理组件把这些调用作为RPC转发给SSim 单元。另一种互动的方式是通过操作系统级别的共享内存。软件单元可以访问被SSim单元映射到共享内存区域的模型输入/输出，如图1b所示。该技术允许与未实现RPC或Modbus的软件进行互动，提供了一个简便的方式开展更为复杂的安全性研究。

多任务操作系统上的实时模拟在实时模拟中，模型在与操作系统的时钟紧密关联的离散时域内运行。也就是说，模拟的模型与实际物理系统模型的运行速度相同。EPIC 使用了装有多任务操作系统的通用PC来运行实时软件模拟单元。尽管优势巨大，选择Simulink Coder创建模拟器也存在不足，它会对模拟的模型施加多种限制，包括模型的执行速度，或是“模拟步骤”。模型的内部动态情况也限制了模拟步骤的可选范围。在选择模拟步骤之前，EPIC研究人员必须验证实时模拟的输出是否尽可能准确地再现了对应的真实世界过程的输出。采用并行运行时，模型在特定计算机上的执行时间取决于模型的复杂度和主机的处理能力。一般情况下，如果模型的执行时间超过了模拟步骤的时间,则无法实现实时模拟。

为了测试EPIC中软件模拟的局限性,我们利用几个物理过程进行了实验(见图2)。本文中我们探讨了小规模的过程（比如基于瑞典马尔默的Sydsvenska Kraft AB电厂构建的Bell和Åström160MW燃油发电厂[1]，以及同样基于真实过程构建的田纳西-伊斯曼化工过程（Tennessee-Eastman chemical process））[9]，虽然Downs和Vogel[9]引起了一些变化来隐藏反应物和生成物的特性。我们在实验中一直使用的铁路系统基于Rios和Ramos提出的列车模型[18]构建，其中考虑了真实交通系统中的多个维度（比如重量、速度、加速、减速以及耗电量）。

然后，我们在EPIC中使用了IEEE的电网系统套件[22]。九辆客车的测试用例为美国西部联合电网（Western System Coordinating Council）的三机九客车系统，而30客车，39客车以及118客车的测试用例则取材于二十世纪60年代初俄亥俄州哥伦布市美国电力公司（American Electric Power）运营的系统的一部分。这些测试用例构成了电力系统社区中历史悠久的真实模型，提供了多种电力系统的配置方案。

实时软件模拟非常适于中小型模型，图2中的数据反映了具体的情况。然而，软件模拟受到CPU速度和模型规模的限制；例如，IEEE118客车系统是一个复杂的模型，其中包括54个频率为50Hz的发电机，以及一个时长约为24ms的最大模拟步骤，或者说最快的模型执行速度。因为模型在2.8GHz CPU上的执行时间为155ms，所以在这种情况下无法进行实时模拟。

实时软件模拟这一局限已是众所周知，但是可从以下几个方面克服这一局限；例如，研究人员可以利用并行处理技术（比如GPU计算）或更为强大的、专门为模拟设计的专用硬件模拟器。然而，这些方法仍然耗资高昂，而且在多模型的环境中，可能会让网络-物理测试床的成本变得无法承受。

实施细节 在意大利伊斯普拉市（Ispra）的欧盟委员会联合研究中心，EPIC的设施由120台PC组成，上面运行了约100台虚拟机，它们通过两层堆叠的网络交换机大规模地连接在一起。另外，还配置了运营商级别的路由器（比如Cisco 6503）及工业控制硬件和软件（比如ABB AC 800M控制硬件，包括与控制服务器相连的Modbus 接口和ABB开发的人机接口软件）作为实验资源。我们还用C#开发了多个软件单元（比如SSim 和代理（Proxy）），然后利用Mono平台把它们移植到了基于Unix的系统，并进行了相关测试。Mono 平台支持C#应用的跨平台部署。

可扩展性和适用性

EPIC具有重建NCI网络和物理维度的能力，这使得它可以提供范围广泛的实验选项来处理关键基础设施。在下文中，我们概括了使用EPIC开展的各种典型实验，并详尽地说明了实验场景：

一般情况下，如果模型的执行时间超过了模拟步骤的时间，则无法实现实时模拟。

典型实验 很多研究人员同时利用EPIC开发、测试和验证一系列的概念、原型和工具（见表2）；您还可以访问EPIC网站http://ipsc.jrc.ec.europa.eu/?id=693获取更多的科学报告和论文。本文中描述的第一个实验度量了各网络参数对针对NCI的网络攻击的影响。我们观察了网络延时、数据包丢失、背景流量和网段划分对具有欺骗性质的网络攻击的影响。该网络攻击由可向过程控制器发送合法指令的一个对手组成。本文说明，虽然通信参数对网络攻击没有明显的影响，但是与物理过程有关的网段划分会产生更有弹性的系统。

第二个实验说明，各项研究有助于验证新提出的保护机制的有效性。使用EPIC后，我们重建了复杂的设置，包括真实的网络、协议、主机和路由器，因此我们拥有了一个真实的环境来验证新奇的异常侦测系统。实验证实，研究人员可以使用EPIC重建真实环境，启动分布式拒绝服务攻击（DDoS），并同时启动对关键基础设施资产的欺骗攻击。这些攻击有助于验证新奇的异常侦测系统是否可以有效地侦测NCI网络和物理维度的异常。

第三个实验重点关注了人工操作员，对网络-物理实验中的重要一环进行了闭环。实验中包括协调一致的网络攻击，攻击中通过阻断通信阻止了对几个变电站的常规远程操作，或者说“负荷减少”。结果，几个变电站的电压出现了明显的下降，显著低于额定运行水平。该实验说明，操作决定可以成为确定完全崩溃或系统生存的决定因素，而且操作员之间的协作可以限制网络干扰的传播。

第四个实验重建了广为人知的2008年YouTube 边界网关协议路由劫持事件[19]，分析了几个假定的场景。我们开发了欧洲互联网骨干网的抽象模型，通过重现真实的流量

数据重建了这一事件。实验结果强调了对边界网关协议路由劫持事件进行快速侦测的工具及其相关机制的重要性，最重要的是，训练有素的操作员能够通过可信的媒介进行通信。

总体来说，这四个实验只代表了多种领域和应用的一小部分。在这些领域和应用中，EPIC已经证明了自己是一个现代的科学工具。不仅如此，它的用途不仅包括破坏性的实验，还可以用于教学和预案方面的活动（比如用于构建网络安全的练习环境）。

说明性的实验在本文中，我们通过探讨ICT中断可能对关键基础设施（比如国家性的电网）的影响说明了EPIC的适用性。我们考虑了网络攻击的假设场景 - 特别是DDoS 攻击- 造成电信服务质量出现明显下降，这种影响还会在关键基础设施中传播。

*实验的设置*据此，我们重建了一个典型的架构，其中电网采用了远程控制的方式（见图3a）。位于运营商经营场所的站点A运行了一个简化的能源管理系统模型（EMS）[21]，以确保电压稳定性。EMS持续不断地用远程的方式监测和调整运行于站点B（比如在变电站中）的电网模型的运行参数。EMS向仿真的控制硬件发送指令，或者通过提供电网模型输入输出的访问途径的代理组件发送指令。彼此间的通信通过Modbus 协议实现。

在类似的研究中，科学界广泛采用了IEEE的电网模型，因为它们（准确地）封装了真实基础设施的基本特性。我们采用了IEEE 39-客车的新英格兰系统，其中包括39个变电站和10台发电机。系统承载的日负荷源于真实数据，[13] 需要EMS的介入来保持电网稳定。

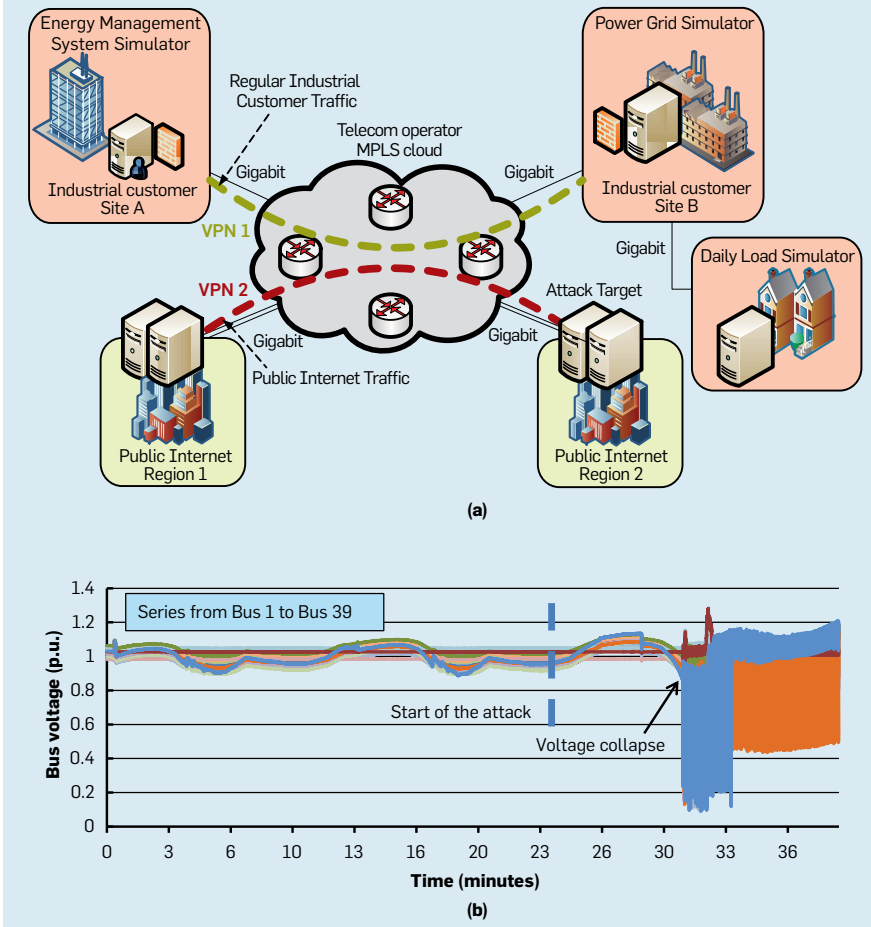
假设在EMS和电网模拟器之间存在真实的通信基础设施，再假设服务提供商使用了多协议标签交换（MPLS）网络；电信运营商使用MPLS协议替代较早的，基于帧中继和异步传输模式的通信实现方式。[12]

使用我们的Emulab 安装包后，我们创建了一个配有四台思科6503（Cisco 6503）路由器的微型MPLS网络，并在上面设置了两个MPLS虚拟专用网（VPN）。VPN 1作为站点A和站点B之间的保护虚拟电路。电信运营商经常采用这种方法来隔离客户的流量。由于电信运营商通过相同的MPLS云经不同路由传送性质不同的流量，我们使用VPN 2来创建连接两个不同“公共”区域的虚拟电路。

*电信中断及其向电网的传播*然后，我们在VPN2中启动了消耗带宽的DDoS 攻击，并测量其对VPN1中电网运营商的虚拟电路的影响。该攻击危害了电网运营商的专用电路。导致EMS失去了对电网的控制，无法发送可以恢复稳定的指令。攻击开始后，电网可以运行约七分钟，无须干预（见图3b）。然而，七分钟后，需要利用在EMS中实现的切负荷算法干预日负荷的变化以保持电网稳定。因为EMS发出的指令无法到达仿真的控制硬件，电网中不同线路分段的电压不可避免地开始出现崩溃。

攻击开始后不久，模型变得极不稳定，呈现了难以映射到现实的巨

图 3网络攻击对关键基础设施的影响：（a）在实验环境中，包含三个物理SSim 单元，一个能源管理系统模拟器，多个攻击者节点以及电信运营商提供的两条虚拟电路；VPN 1 = 电网运营商的专用电路，VPN 2 = 公共互联网；以及（b）对电压稳定性的影响。



大震荡。基于模拟的研究的主要局限性在于研究人员只能在模型的范围内推理。然而，电压崩溃明显说明电网不稳定，这可能会促使运营商重新建设整个电网。对于我们与EMS有关的安全性研究而言，其足以验证攻击者可让系统超出正常的运行限值。如果在实验中超出了这些限值，那么研究人员必须用类似的方式拓展物理系统的模型，使之覆盖极端和不稳定的条件，或是通过真实的物理设备拓展网络-物理测试床，此时假定拓展是可行的，在经济上具有成本效益，而且安全。

审视现实 大多数电信运营商对独立的VPN之间的干扰进行了限制；例如，在MPLS网络中配置服务质量（QoS）后，对公共互联网的攻击几乎不影响其他电信客户的专用流量。在启用MPLS云中配有数据包优先级的QoS（在工业通信中，也使用了这一特征实现数据包优先级）后，通过运行与EMS有关的实验，我们验证了这一观点。唯一能测出的影响是，数据包的往返时间略有增加（增加了1ms至2ms）。依照IEEE 1646-2004变电站自动化用通信延时标准，这种延迟是可以接受的。IEEE 1646-2004规定，高速的信息必须在2ms至10ms的范围内送达。

然而，政策和规范中规定的上述措施并没有强制力。我们开展的EMS相关的实验说明，如果没有实施这些措施，会造成严重的风险。我们的实验还强调了ICT中断对各类物理系统的潜在威胁。不仅如此，通过设计和开展基于真实事件的实验，我们可以探讨各种假设场景。例如，我们调研了2004年发生在罗马的事件。该事件影响了通过公共电信网络管理的，受远程控制的电网。[4] 由于水管破裂，电信运营商的服务器机房被淹，造成关键硬件短路，引发远程站点之间的通信中断。电网运营商无法监视或控制远程站点。幸运的是，这些干扰均没有造成损害，所以电网依然稳定。然而，我们利用EPIC开展的实验表明，如果产生的能量与消耗的能量之间的平衡发生变化，可能会对电网造成严重的影响。2004年罗马市的人口数量为250万。上述事件可能会引起全城停电，并影响其他的关键基础设施（比如交通和医疗保健）。

结论

把基于Emulab的测试床与实时软件模拟器结合在一起后，赐予了EPIC一个新方法来研究包含多个异构NCI的网络安全。我们可以把EPIC看成一类新的科学工具 - 网络-物理测试床 - 适于评估物理基础设施面临的网络威胁。EPIC支持研究人员在包含异构系统以及众多相互依赖的关键基础设施的领域（比如交通、化工制造和电网）内开展有趣的研究；有关详细情况，请访问http://ipsc.jrc.ec.europa.eu/?id=691。

参考资料

1. Bell, R. and Åström, K. *Dynamic Models for Boiler-Turbine Alternator Units:Data Logs and Parameter Estimation for a 160MW Unit.Technical Report TFRT–3192.*Lund Institute of Technology, Lund, Sweden, 1987.
2. Benzel, T., Braden, R., Kim, D., Neuman, C., Joseph, A., Sklower, K., Ostrenga, R., and Schwab, S. Experience with DETER:A testbed for security research. In *Proceedings of the International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities* (Barcelona, Mar. 1–3).IEEE, New York, 2006, 379–388.
3. Bergman, D.C., Jin, D., Nicol, D.M., and Yardley, T. The virtual power system testbed and inter-testbed integration.In *Proceedings of the Second Conference on Cyber Security Experimentation and Test* (Montreal, Aug. 10–14).USENIX Association, Berkeley, CA, 2009, 5–5.
4. Bobbio, A., Bonanni, G., Ciancamerla, E., Clemente, R., Iacomini, A., Minichino, M., Scarlatti, A., Terruggia, R., and Zendri, E. Unavailability of critical SCADA communication links interconnecting a power grid and a telco network.*Reliability Engineering and System Safety 95*, 12 (Dec. 2010), 1345–1357.
5. Charette, R. IT Hiccups of the week:Southwest Airlines computer failure grounded all flights.*IEEE Spectrum Risk Factor Blog* (June 2013); http://spectrum.ieee.org/riskfactor/computing/it/it-hiccups-of-the-week-southwest-airlines-computer-failure-grounded-all-flights
6. Chen, T. and Abu-Nimeh, S. Lessons from Stuxnet. *Computer 44*, 4 (Apr. 2011), 91–93.
7. Chertov, R., Fahmy, S., and Shro, N.B.Fidelity of network simulation and emulation:A case study of TCP-targeted denial-of-service attacks.*ACM Transactions on Modeling and Computer Simulation 19*, 1 (Jan. 2009), 4:1–4:29.
8. Chunlei, W., Lan, F., and Yiqi, D. A simulation environment for SCADA security analysis and assessment.In *Proceedings of the 2010 International Conference on Measuring Technology and Mechatronics Automation* (Changsha City, China, Mar. 13–14).IEEE, New York, 2010, 342–347.
9. Downs, J. and Vogel, E. A plantwide industrial process control problem.*Computers & Chemical Engineering 17*, 3 (Mar. 1993), 245–255.
10. Duggan, D. *Penetration Testing of Industrial Control Systems.Technical Report SAND2005-2846P.*Sandia National Laboratories, Albuquerque, NM, 2005.
11. Hahn, A., Ashok, A., Sridhar, S., and Govindarasu, M. Cyber-physical security testbeds:Architecture, application, and evaluation for smart grid.*IEEE Transactions on the Smart Grid 4*, 2 (June 2013), 847–855.
12. IBM and Cisco.Cisco and IBM provide high-voltage grid operator with increased reliability and manageability of its telecommunication infrastructure. IBM Case Studies, 2007; https://www.cisco.com/web/partners/pr67/downloads/756/partnership/ibm/success/terna_success_story.pdf
13. Manera, M. and Marzullo, A. Modelling the load curve of aggregate electricity consumption using principal components.*Environmental Modeling Software 20*, 11 (Nov. 2005), 1389–1400.
14. McDonald, M.J., Mulder, J., Richardson, B.T., Cassidy, R.H., Chavez, A., Pattengale, N.D., Pollock, G.M., Urrea, J.M., Schwartz, M.D., Atkins, W.D., and Halbgewachs, R.D. *Modeling and Simulation for Cyber-Physical System Security Research, Development, and Applications.Technical Report SAND2010-0568.* Sandia National Laboratories, Albuquerque, NM, 2010.
15. Nai Fovino, I., Masera, M., Guidi, L., and Carpi, G. An experimental platform for assessing SCADA vulnerabilities and countermeasures in power plants. In *Proceedings of the Third Conference on Human System Interactions* (Rzeszow, Poland, May 13–15). IEEE, New York, 2010, 679–686.
16. Nan, C., Eusgeld, I., and Kröger, W. Analyzing vulnerabilities between SCADA system and SUC due to interdependencies.*Reliability Engineering & System Safety 113* (May 2013), 76–93.
17. Queiroz, C., Mahmood, A., and Tari, Z. SCADASim:A framework for building SCADA simulations.*IEEE Transactions on Smart Grid 2*, 4 (Sept. 2011), 589–597.
18. Ríos, M.A. and Ramos, G. Power system modelling for urban mass-transportation systems.In *Infrastructure Design, Signaling and Security in Railway*.InTech, Rijeka, Croatia, 2012, 179–202.
19. RIPE Network Coordination Centre.*YouTube Hijacking:A RIPE NCC RIS Case Study*, 2008; http://www.ripe.net/internet-coordination/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study
20. Siaterlis, C., Garcia, A., and Genge, B. On the use of Emulab testbeds for scientifically rigorous experiments.*IEEE Communications Surveys and Tutorials 15*, 2 (Second Quarter 2013), 929–942.
21. Tuan, T., Fandino, J., Hadjsaid, N., Sabonnadiere, J., and Vu, H. Emergency load shedding to avoid risks of voltage instability using indicators.*IEEE Transactions on Power Systems 9*, 1 (Feb. 1994), 341–351.
22. University of Washington.*Power Systems Test Case Archive.*Electrical Engineering Department, Seattle, 2012; http://www.ee.washington.edu/research/pstca/
23. U.S.Department of Energy.*National SCADA Test Bed.* Washington, D.C., 2009; http://energy.gov/sites/prod/files/oeprod/DocumentsandMedia/NSTB_Fact_Sheet_FINAL_09-16-09.pdf
24. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. An integrated experimental environment for distributed systems and networks.In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation* (Boston, Dec. 9–11).USENIX Association, Berkeley, CA, 2002, 255–270.
25. Yardley, T., Berthier, R., Nicol, D., and Sanders, W. Smart grid protocol testing through cyber-physical testbeds.In *Proceedings of the Fourth IEEE PES Innovative Smart Grid Technologies Conference* (Washington, D.C., Feb. 24–27).IEEE Power and Energy Society, NJ, 2013, 1–6.

Christos Siaterlis（christos.siaterlis@jrc.ec.europa.eu）是意大利伊斯普拉市欧盟委员会联合研究中心公民保护和安全研究所的项目主任。

Béla Genge（bela.genge@ing.upm.ro）是罗马尼亚穆列什县Petru Maior University of Tîrgu Mures大学信息学系Marie Curie博士后研究员和委员。

译文责任编辑：陈贵海

DOI:10.1145/2500874

原子级一致的内存服务可为动态场景提供弹性。

作者：PETER MUSIAL，NICOLAS NICOLAOU，
ALEXANDER A. SHVARTSMAN

为动态网络实现分布式共享内存

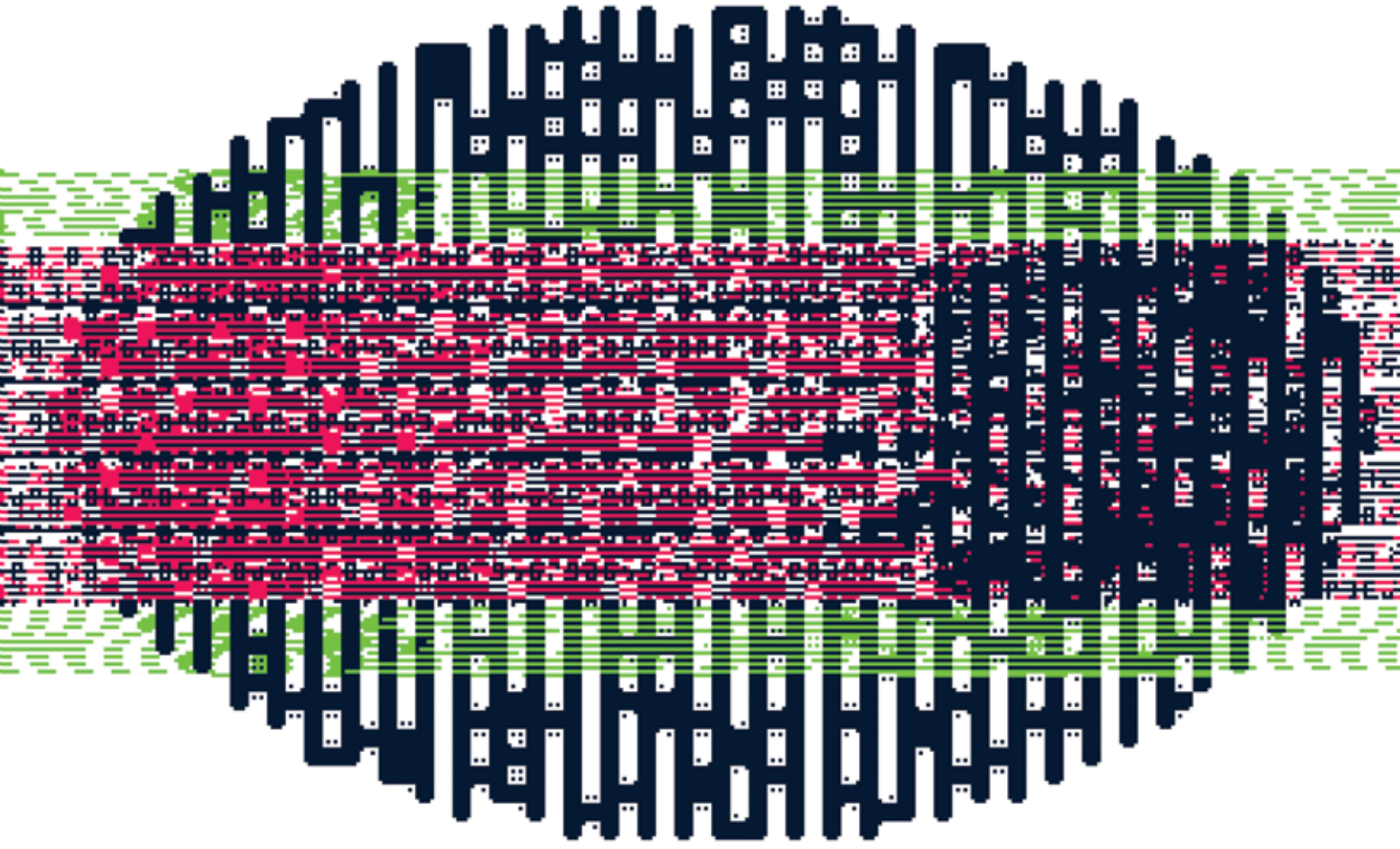
*阅读、“写作”和“计算”（READING, ‘RITING, 和‘RITHMETIC）*是隐藏在人类智力活动背后的三个R。如果说这三个R也是现代计算技术的庄严基石，人们也不会觉得意外。实际上，图灵机和冯·诺依曼计算机模型均通过读、写和计算运行。所有实用的单处理器计算机也都以执行依照三个R构建的活动为基础。随着网络技术的进步，通信成了另一个重要的系统性活动。然而，站在高度的抽象水平来看，用读、写和计算的概念思考仍就明显更符合习惯。

虽然人们难以设想没有通信的分布式系统 - 比如实现万维网的分布式系统，但是我们通常可以想到基于浏览器的应用需要通过获取（也就是读）数据、执行计算和存储（也就是写）结果的方式运行。在本文中，我们探讨了如何存储分布式系统中共享的可读写数据，重点讨论了在动态场景中提供弹性和一致性的实现。动态场景系指在由计算机及连接计算机的网络组成的底层分布式平台中，各个系统会受到扰动的影响。这些扰动包括独立的计算机出现的故障，动态改变参与系统的计算机集合，以及通信介质中的故障和延迟。

共享存储服务是信息时代大多数系统的核心。本文考查的共享内存系统提供了支持两种访问操作的对象：*读*，获取对象的当前值；以及写，利用新值替换对象的旧值。这些对象通常被称为*寄存器*。虽然本文没有探讨更为复杂的对象语义，比如事务或整合后的读-改-写操作，但是本文阐述了任何分布式存储系统均需解决的，共同的实现挑战。假设存储系统通过中心服

» 重要见解

- **动态共享内存（DSM）系统支持使用读写操作访问的对象，还能在底层分布式平台出现扰动的情况下保证一致性和可用性。**
- **开发人员可以使用DSM实现使用共享内存范式的分布式系统，避免在底层的消息传送设置、异步或故障处理中花费过多精力，从而把工作聚焦于系统特性。**
- **DSM系统采用了对对象副本集合进行运行时透明重构的机制，并支持以从存储局域网的节点到自组织网络中的移动设备等设施作为基础的多种副本实现。**



图片由 IGNOTUS THE MAGE 提供

务器的方式实现。服务器接受客户端的请求，在它的数据对象上执行操作，然后返回响应。虽然概念上比较简单，但是这种方式已经面临了两个重大的问题。首先，该中心服务器是一个性能瓶颈。其次，该服务器会导致单点故障。客户端的数量增加后，这种实现的服务质量会迅速下滑。不仅如此，如果服务器崩溃，服务会变得不可用（设想一下，如果在线新闻服务使用中心服务器来实现，会存在多大的不足）。因此，最重要的一点是系统必须可用。这意味着，即使在系统规范规定的范围之内出现故障，系统也必须提供服务。举例来说，系统必须能够屏蔽某些服务器和通信故障。系统还必须支持多并发访问，而不会造成性能出现不合理的下降。保证可用性的唯一方法便是*冗余*，也就是说，利用多台服务器，然后把各对象的内容复制到各台服务器上。不仅如此，还必须把副本复制到多个分散且不同的异地网络中，这样系统便能够屏蔽某些数据服务器子集的失联或故障。

保证数据的生存时间也极为重要。存储系统或许能容忍某些服务器的故障，但是经过一段较长的时间后，由于各台服务器都会出现宕机，再加上计划内的升级，可以想象可能需要更换所有的服务器。另外，在移动的场景中，例如，搜救或军事行动中，可能需要提供把数据从一个服务器集合迁移到另一个服务器集合上的能力，以便数据能够按需移动。无论我们的关注点是数据的生存时间或是移动性，存储系统都必须提供无缝的运行时数据迁移：人们无法停止世界来重构系统以应对故障和不断变化的环境。

复制面临的一个主要问题是一致性。系统怎样找出复制的对象的最新值？采用中心服务器的实现不存在这种问题：该服务器一直拥有最新值。在采用复制的实现中，人们可能会试图查询所有的副本以找出最新值，但是这种方法的开销相当高，而且无法容错，因为它假定可以访问所有的副本。无论在什么情况下，分布式内存服务的客户端都应该不需要关注这类实现问题。客户端应该期望看到一种只有一个单拷贝的对象的假象，这种对象会序列化所有的访问，使得每次读操作返回上次写操作的值，且该值至少应与之前的任何读操作所返回的值一样新。从更普遍的角度来说，从外部观察到的对象行为应该与对象的抽象序列化数据类型一致。而且，在使用此类对象开发应用时，客户端必须能够依赖对象的抽象数据类型。一致性的概念可形式化为*原子性*[28] 或为等效的*线性一致性*。[25]

对于原子性是一致性中最方便的概念这一观点，虽说没人会觉得有问题，但是我们发现，主要是在受效率因素的驱动后，业界也提出和实现了较弱的概念。举例来说，

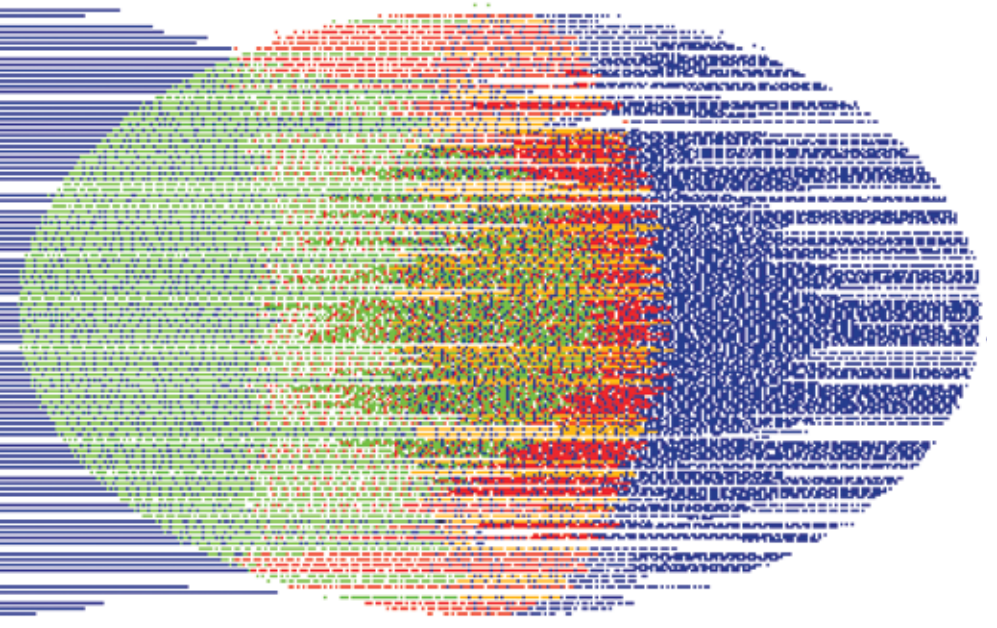
在包含多处理器内存系统域中，出现一些不大符合直觉的一致性定义，仓促地证明了一个坊间观点，“虽说没人能弄清这些一致性模型，但是内存访问很快。”原子性提供了相当强的保证，但与较弱的一致性保证相比，实现原子性的开销更高。[4]在可分区的网络设置中，也需要考虑弱一致性。Eric Brewer猜想[7]，分布式存储系统不能同时提供*一致性*、*可用性*和*分区容错性*；人们称之为“CAP猜想”，随后CAP猜想被形式化为一个定律。[23]因此，某些场合下需要考虑较弱的一致性模型。[8]不过，我们仍然认为，提供简单和符合直觉的原子一致性相当重要。

这样的情况让人回想起编程语言发展的初期阶段。那时人们争论说，汇编语言更好，因为汇编语言能生成更有效的代码。或是让人回想起图形用户界面的初期。那时人们争论道，命令行界面更好，因为它们消耗的资源更少。可以想象，对于原子性的争论也会逐渐散去。如果在设置中需要关注网络分区，那么可以考虑下列两种方法：如果分区是间歇性和短暂的，则可以把它们当成（较）长的网络延时进行处理，把修复问题的任务委托给更底层的通信服务。在其他情况下，如果分区是永久的，强一致性仍然可以通过限制对主分区的服务、然后在分区合并时进行相应整合的方法予以保证，[30]2008年ACM A.M.图灵奖得主最近发表的主题演讲中也提到了相关的内容。图灵奖得主Barbara Liskov提出，虽然原子性开销不小，但如果我们不确保原子性，就会让开发人员头痛。

有趣的是，某些现有的存储系统提供了实现原子级*读-改-写*操作的数据访问原语。这类访问原语比本文探讨的独立的*读*和*写*的原语要强得多。实现这种类型的一致性开销很高，它实质上要求实现原子级更新。在实际场景中，这种原子级更新或是通过把部分系统归约成一个*单一写入器*模型（例如，微软的Azure[9]），或是依靠时钟同步硬件（谷歌的Spanner[13]），或是依赖复杂的机制（如向量时钟）解决事件的顺序问题（亚马逊的Dynamo[15]）。本文对原子级读/写存储的阐述说明了所有分布式存储系统都面临的共同挑战。

分布和一致性

此处，我们描述了一个通用分布式场景用于实现一致的共享内存服务。



构建分布式平台的模型 我们构建的系统模型为互联的计算机或节点集合，它们之间通过发送点对点消息进行通信。每个节点都有一个唯一的识别符（比如IP地址）和本地存储，且能执行本地计算。计算的任何步骤都可能崩溃，导致节点出现故障。崩溃的节点停止了运行：它无法执行任何本地计算，无法发送任何消息，而发送给它的任何消息也无法送达。

系统是异步的，节点无法访问全局时钟或同步机制。这意味着，节点的相对处理速度可以是任意的，而且节点不知道执行本地计算时所需时间的上限。消息的延迟也可能是任意的，而且节点不知道消息延迟的范围（虽然可能存在此类范围）。因此，算法可能无法对全局时间或延迟做出假设，因为不知道各节点的处理速度和消息延迟。

我们假设，在传输的过程中可以重新安排消息的顺序，但是，无法同时破坏、复制或生成消息。如果消息被接收了，那么它必须是之前发出的消息。消息不会丢失，而且消息丢失在模型中可以构建为长延迟（我们没有关注用于构建更可靠的通讯服务的技术。例如，使用重发或流言传播协议）。

我们把这种分布式网络系统分为下列的静态或动态两类.在静态网络系统中，参与节点的集合是固定的，每个节点或许能知道其他每一个参与者的身份；崩溃（或者自愿离开）可能会从系统中移除节点。在动态网络系统中，节点的集合可能不受限制，而且由于崩溃、离开或者新节点的加入，参与节点的集合可能在一段时间后完全不同。在这一点上，我们没有对动态场景中的故障规模或“流失”做出假设。直到在提到特定的解决方案后，我们才开始进行相关讨论。不管动态的情况如何，我们都要求保持数据的一致性。计算介质中的

扰动可能会降低我们所考虑的内存服务的性能。不过，在特定假设的约束下，内存访问操作保证能够终止。例如，静态的内存服务保证，在多数副本是活跃的且网络延迟受限时，操作能够终止。动态内存服务放松了静态的多数性假设。作为替代，假设动态变化的副本子集（例如，动态仲裁）在某些时间段是活跃的。即使没有满足这些假设，该类服务仍能保证原子性，不过，某些操作可能会很慢，或是可能不会终止。

总体来说，我们认为参与的节点是“良民”，它们在能够合作的时候乐意合作，既不会蓄意执行恶意操作，也不会偶然执行不正确的计算。因此，本文不会探讨处理恶意行为的技术。不过，我们引导有兴趣的读者阅读研究参与者可能实施恶意行为的相关文献，例如Rodrigues等人[35]以及Martin和Alvisi[34]的著作。

*分布式共享内存和一致性*分布式共享内存服务仿真了由在网络平台上运行的可读写对象（通常称为寄存器）组成的共享内存空间。网络平台由通过传递消息进行通信的分布式网络节点组成。服务的实现使用了副本来保证对象的生存能力和可用性，但是对客户端来说，服务屏蔽了这些细节。每个对象的内容被复制到多台服务器或副本主机上。客户端调用对象上的读写操作，其中执行读操作的客户端称为读取器，执行写操作的客户端称为编写器（客户端可能同时为读取器和编写器）。

为了响应客户端的请求，服务调用了一个协议与副本主机进行通信。该协议确定了内存系统的一致性的保证措施。在原子级一致性的定义中，把执行每一步操作的时间“压缩”至操作的调用和响应之间的一个选定串行化点，要求按照串行化点编排的操作顺序保留它们的

复制面临的一个主要问题是一致性。系统怎样找出复制的对象的最新值？

实时顺序且由此产生的对象行为与其序列说明一致。特别的是，如果在一次写完成之后调用读，那么服务保证读的返回值是此次写的值，或是读之前发生的后续其他写操作写入的值。另外，如果在一次读完成之后调用读，那么服务返回相同的值，或者是比上次读“更新”的值。（我们在附录中用更为正式的方式阐述了这点，您可以从ACM数字图书馆获取该附录。）正是因为这些自然的属性，原子性成了最方便、最符合直觉的一致性保证。

Lamport[28]引入了原子寄存器的概念（他还定义了较弱的概念，比如安全性和规律性）。Herlihy和Wing[26]提出了称为线性一致性的等价定义，也把原子性拓展到了任意数据类型上。既然原子性是一致性中最强的概念，提供了最有用的访问语义，后面我们就把重点放在原子级的共享内存系统上。

构件：静态网络系统中的共享内存

为静态场景设计的算法必须能够容忍某些动态行为，比如不同步、瞬时失效以及在某些限值范围内的永久崩溃。在此我们阐释了用于上述场景的共享内存服务，其中的两点原因如下：为下一节中更动态的服务做铺垫；以及关注实现原子性的方法，因为几个动态服务中也使用这个方法。Chockler等人[12]对静态场景中共享内存的实现做了首个综合性的调查。在他们的著作中，某些副本访问协议可以作为动态实现的构件。不过，为静态场景设计的算法不能直接使用，因为他们无法处理动态变化的副本集。其中的一种算法是Attiya，Bar-Noy和Dolev[3]提出的，具有开创意义的原子级共享内存的实现，人们通常把该算法称为ABD算法；该项研究赢得了2011年的迪科斯彻奖（Dijkstra Prize）。

该算法实现了原子级内存，其中的副本帮助服务实现了容错和可用性。系统可以容忍f个副本节点的崩溃，只要副本的数量n 满足$n > 2f$。我们的阐释中还包括了Lynch和Shvartsman[32]提出的对原算法的扩展。

每个副本主机i 维护了寄存器的本地值$value_i$，以及与副本关联的一个标签$tag_i = \langle seq, pid \rangle$，其中$seq$指序号，$pid$指执行该值写入操作的节点标识符。标签按字典顺序进行比较。每次新的写入会配有一个唯一的标签，其中发起写入的节点id被用于打破关联。实际上，这些标签确定了写操作的顺序，因此也确定了其后读操作的返回值。

读操作和写操作较为相似，各自的实现都有两个阶段：Get（取值）阶段查询副本获取信息，Put（赋值）阶段把信息传播给各个副本。每个阶段的协议均确保特定多数参与了信息互换：首先消息被发给了所有的副本主机，然后收集回复，直到特定多数的副本响应为止。让我们回顾一下，因为$n > 2f$，所以由没有崩溃的副本组成的多数总会存在。因此，在一个单一的通信回合后，每个阶段终止；在两个通信回合后，任何操作终止。

该实现的正确性，也就是原子性，基于下列事实：对于成对的先后操作，在第一次操作的Put 阶段以及第二次操作的Get阶段中，至少包含了一个正确的副本；这就保证了第二次操作时“看到”的值总是至少与之前最近的一次操作的值新旧程度相同。我们指出，与数个多数一同使用的唯一属性是，任何两个多数相交。如果等待多数的处理太消耗任务，人们可以使用仲裁系统（quorum system）作为替代。[32,38]（详细分析和完整的伪码在附录内，您可以从ACM 数字图书馆获取该附录。）

共享内存服务的客户端应该想形成一种错觉，看到一个序列化所有访问的单拷贝对象。

动态网络系统中的共享内存仿真

在此，我们阐述了在更为动态的系统中提供一致的共享内存的几个方法，也就说，在这些场景中，节点不仅会崩溃，还会自愿离开，而且还可能会有新节点加入服务。总体来说，对象副本的集合可能随时间发生明显的演化，最终会移植到完全不同的副本主机集合上。在这种场景下，不能直接使用ABD算法，因为该算法依赖于多数原始副本主机一直可用这一条件。为了在动态场景中使用类似ABD的方法，人们必须提供一些管理副本主机集合的措施，以确保读取器和编写器接触合适的主机集合。

值得注意的是，处理动态场景和管理节点集合并未直接解决如何提供内存服务的一致性的问题。与此相反，这些问题代表了在动态分布式计算域中存在的更广泛的挑战。实践说明，一致的共享内存服务有时可以通过分布式构件搭建，比如设计用于管理参与节点集合的构件，用于提供合适的通信原语的构件，以及用于在动态分布式场景中达成一致（共识）的构件。Aguilera 等人[1]的教程中涵盖了其中的几个话题。

首先，我们探讨共识问题，因为通过建立操作顺序的共识可为实现原子级内存服务提供一个自然的基础，而且在原子级内存的实现中，其他方式也用到了共识。然后我们探讨组通信服务（GCS）解决方案，其中使用了强通信原语（比如全序广播）编排操作的顺序。最后，我们重点关注了把ABD算法的思想扩展到动态场景中的方法，其中包含对不断演化的副本主机集合的显式管理。

*共识*在分布式场景中达成一致是计算机科学中的一个根本问题。人们把分布式场景中的一致问题称为共识问题。例如，一组进程集合需要就一个值达成一致，但节点提

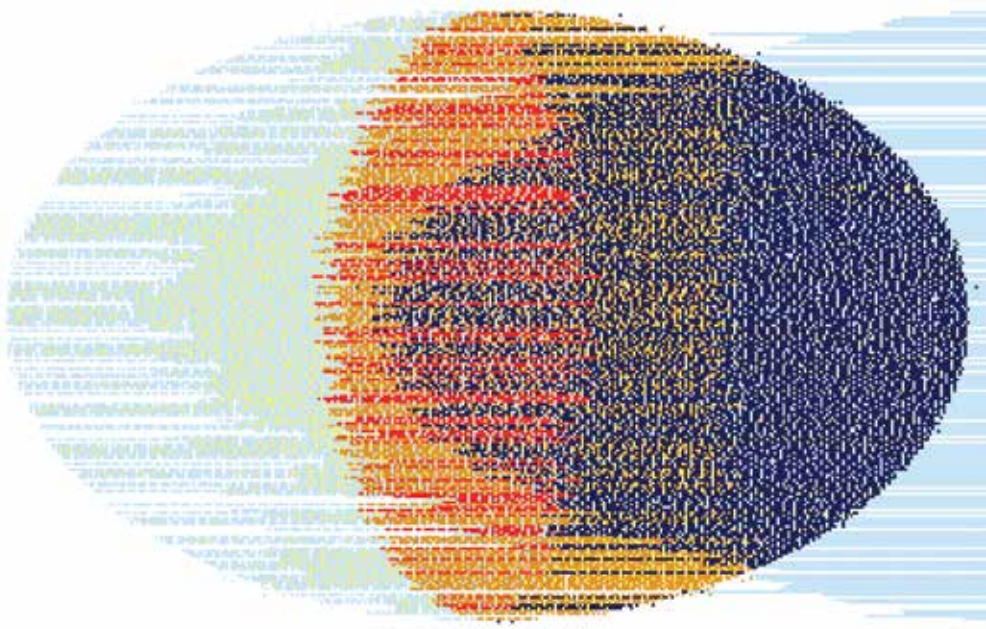
出了几个备选值：所有的解决方案都必须包含下列特性：*一致*：不存在使用两个进程决定不同的值；*有效性*：决定的值由某个进程提出；*终止*：所有正确的进程做出一个决定。共识是设计分布式服务时使用的一个强大的工具[33]。然而，在异步系统中，共识是一个臭名昭著的难题，因为如果出现单一的进程崩溃，终止就无法保证；[19]因此，必须小心地使用共识。（解决共识问题的方法之一是引入能提供系统中节点信息（可能有限的信息）的“故障检测器”[10]。）

通过让参与者使用共识对所有操作的全局全序达成一致，共识算法可被直接用于实现原子级数据服务。[29]无论人们选择哪种特定的共识实现，此方法的正确性（原子性）都可以得到保证。不过，对底层平台特征的理解可以引导人们选择提升系统性能的实现（Lynch[33]创作了实现方面的佳作）。然而，为每个操作使用共识的开销很大，特别是在扰动可能会延迟甚至是阻止终止的情况下。因此，在使用共识时，必须避免与独立的内存操作同时调用共识，必须让操作独立于共识的终止。

我们注意到，达成共识的难度比实现原子级读/写对象的难度要高很多。特别是两个或多个进程的共识无法通过使用原子级读/写寄存器解决。[31]

*组通信服务*分布式系统的另一个最重要的构件是让位于网络不同节点的进程作为一个组来共同运行的组通信服务（GCS）[5]。各进程通过使用GCS组播服务向组内的所有成员发送消息来实现这一点。GCS提供了多种手段保证消息传递的顺序和可靠性。

GCS的基础是*组成员服务*。每个进程每次都拥有组的唯一视图，其中包括一个组成员的进程列表。视图会随时间变化，而且不同进程的视图可能不同。GCS方法引入的另一个重要概念是虚同步，其中最关键的需求是：如果进程通过两个连续的视图走到了一起，那么进程在这些视图之间会提交相同的消息集合。这可让接收者依照消息、会员集合以及应用规定的规则采取协调一致的行动。[5]



GCS提供了一种方法来实现动态网络中的共享内存。举例来说，可以通过在视图同步的GCS[18]之上架构全局全序组播服务实现上述共享内存（其中，对与每个视图关联的消息设置了全序，每个参与者会接收该序列的前缀）。这种有序的组播用于为内存访问操作设置顺序，实现原子级内存。该解决方案的主要缺点是，在大多数GCS实现中，生成一个新的视图需要相当长的时间；在视图生成时，客户端的内存操作会被延迟（或取消）。新视图的生成通常会涉及几个回合的通信。不仅如此，在典型的GCS实现中，即使只有一个节点崩溃，性能也会下降。在内存服务中，人们希望确保在重构时读写操作仍能取得进展。另外，应该能容忍一定数量的故障而不造成性能下降，这点也相当重要。

另一个方法是将GCS与ABD算法集成。例如，De Prisco等人[14]描述的动态主配置GCS通过在各配置内使用Attiya[3]的技术实现了原子级内存。本文中，配置把组视图与仲裁系统结合在一起。由于系统的动态性质，或是由于需要一个不同的仲裁系统，配置可能发生变化。与其他基于GCS的解决方案类似，读写操作在重构时会出现延迟。最后，任何新配置都必须符合与之前的配置有关的某些交集属性。这影响了重构的灵活性，要求在达到所需的配置前进行中间配置变更。

Birman[6]阐述了用于动态服务复制的一般方法论。这种重构模型统一了虚同步方法与状态机复制，恰如在Paxos[29]描述的共识解决方案中使用的方法一样。

*DynaStore*算法DynaStore[2]是一种多编写器/多读取器对象用动态原子级内存服务的实现。它集成了ABD算法，允许在不使用共识的情况下进行副本主机集合的重构。

参与者首先获得了一个默认的本地配置，即某种副本主机的基本集合。该算法支持三种类型的操作：*读*、*写*和*重构*。读写操作包括两个阶段，如果不考虑重构，那么协议与ABD类似：它使用了副本的多数概念，其中每个副本维护了对象的值及其关联的标签。

如果参与者希望改变当前配置，它可以使用重构（reconfig）操作，并向其提供变更的增量集合，例如，用（+, *i*）形式的元素说明增加了副本主机*i*，用（–, *j*）说明移除了主机*j*。*重构*的实现包含两个阶段（如下所述），使用了分布式弱快照服务以通过更新原语的方式广播本地发起的变更，并通过扫描原语的方式获取配置中其他成员提交的变更。快照本身不是原子级的，因为它既不是全局有序更新，也无法保证扫描能反映在扫描前完成的所有更新。尽管如此，快照服务足以建立一个在本地保存的特定的*有向无环图*（DAG），以之作为各参与者状态的一部分。图的顶点对应可以通过变更生成的配置，变更则对应边。

*重构*的实现涉及遍历该DAG，反映已变更的配置的可能序列。在每次遍历中，DAG可能会被修正，以再次生成不同主机就相同配置提交的多次变更。多数参与主机没有被移除，也没有崩溃这一假设确保了在DAG图中存在一条路径，且保证这条路径为通过所有主机的共同路径。有趣的是，主机本身并不知道这条共同路径，不过，遍历所有路径可确保遍历了共同路径。在到达汇点后，遍历终止。另一个假设是，重构的数量有限，可确保终止。

现在，让我们回到*重构*的两阶段结构。第一个阶段的目标与ABD的Get 阶段类似：发现用于标识对象的最新的值-标签对。第二个阶段的目标与ABD的Put 阶段类似：把最新的值-标签对发送给副本主机中的数量合适的多数。它们之间的主要区别在于，在执行重构的两个阶段的过程中，在对配置应用增量变更的同时，还要发现其他参与者提交的变更。这实际上“引导”了可能存在的新配置。鉴于所有工作是通过遍历所有可能的路径来完成 - 所以配置-在DAG中确保共同路径被遍历了。

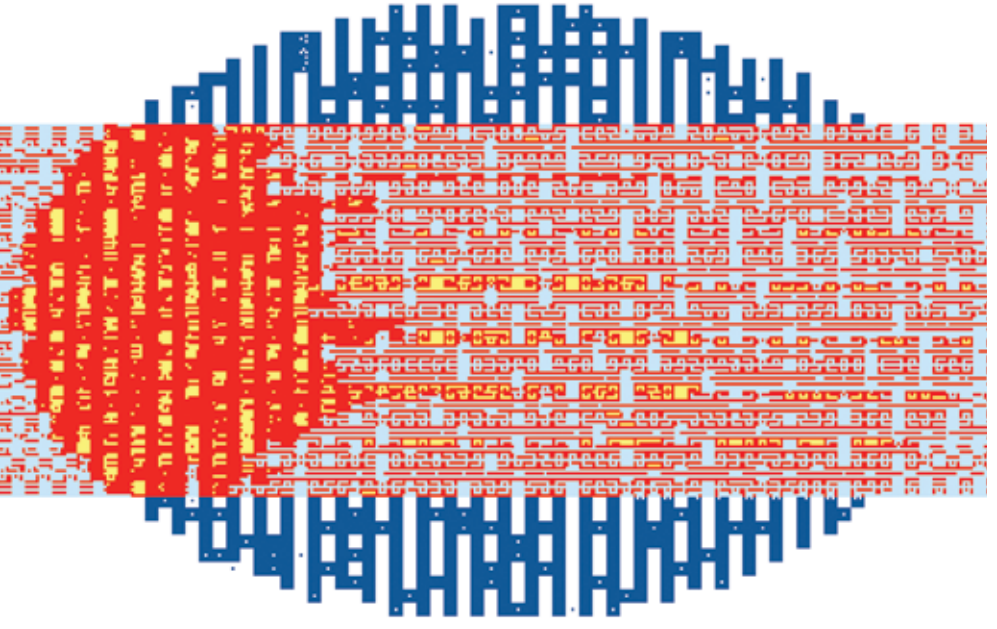
最后，我们说明了读写操作的更多细节。在*重构*实现后，执行*读*，其区别如下：（a）配置变更的集合是空的；以及（b）发现的值返回给了客户端。在*重构*实现后，执行*写*，其区别如下：（a）变更的集合是空的；（b）在第一阶段完成后，生成了一个更大的新标签；以及（c）新的值-标签对在第二阶段中传播。请注意，虽然*读* 和*写*中配置变更的集合是空的，上述两种操作均有可能发现其他地方提交的变更，这有助于引导修正后的配置。对于*重构*，稳定的多数可确保无任何东西阻拦协议取得进展，而且有限数量的重构保证可以终止。

我们觉得有必要重申，DynaStore的实现没有包含用于重构的一致性协议。在另一方面，由于重构通过添加和移除单独节点实现，与通过使用一个配置完全替换旧配置进行系统演进的方法相比，此方法的开销可能较大。所以，*读* *写*操作的延迟更依赖于重构的速度（后面讨论DynaDisk时，我们会谈到这一点）。最后，为了确保终止，DynaStore假定重构最终会慢慢减少直至消失。

*Rambo 框架*Rambo 是一个支持多编写器/多读取器对象的动态内存服务；[24]Rambo是可重构的基本对象用原子级内存（Reconfigurable Atomic Memory for Basic Objects）的缩写。该算法支持多种配置，每种配置由副本主机集和其上定义的仲裁系统组成；它支持重构，可通过重构来替换配置。值得注意的是，仲裁配置可在随时安装，而且从截然不同的配置中产生的各种仲裁之间不需要包含非空交集。该算法确保所有执行均具有原子性。

在没有重构的静止期中，该算法的运行方式与ABD算法类似[3]（Lynch 和Shvartsman[32]进行了概括）：在两个阶段中，每个阶段都涉及与当前配置中的一个完整仲裁进行互动。新的参与者通过向至少一个现有的参与者发送握手消息来加入服务（不涉及任何重构）。

任意一个参与者都可能随时崩溃。为了支持服务的长期运行，人们可以重构仲裁配置。重构与任何正在进行的读写操作并发执行，且不直接影响此类操作。另外，多个重构可以并发进行。重构涉及两个解耦协议：通过名为*Recon*的组件



引入新配置，接着更新至新配置，然后对废弃的单个或多个配置进行垃圾回收。

现在，我们来详细探讨下这种情况。每个参与者维护的主要数据结构是配置映射，或者说*cmap*，其中保存了一连串的配置。在这个映射中，对于节点*i*，$cmap_i(k)$ 指配置号*k*。

在Recon 引入新配置，或对旧配置进行垃圾回收后，该序列会发生演进。对于各*cmap*中的内容，参与者可能有不同的视图。不过，Recon 总是会发出一个唯一的新配置*k*，并将其在每一个$cmap_i(k)$中进行保存。具体的过程如下：假设任意节点*i*是最新已知配置$c = cmap_i(k-1)$的成员，则该节点可随时提出新的配置。通过在*c* 的成员之间执行共识机制，可使不同的配置提议取得一致（举例来说，此时可以使用 Paxos[29]的某个版本实现共识）。虽然共识的执行过程可能会相当慢——事实上某些情况下它甚至无法终止——但这不会使读写操作出现延迟（只要在操作过程中，至少有一组用于相关配置的仲裁集没有受到影响）。请注意到，新配置的成员可能知道，也有可能不知道对象的最新值。*解耦更新* 协议有责任对旧配置进行垃圾回收，并把对象的信息传播给本地已知的最新配置。在此，两阶段的算法首先告诉每个较旧配置的仲裁存在新配置，并获取最新的对象信息，然后再把该信息传播给新配置的仲裁，并移除废弃的配置。

如果重构很快，或是配置更新延后，参与者的*cmap*中可能会有多个活动（尚未被垃圾回收）的配置。在这种情况下，实现读写操作的两阶段协议的行为如下。在第一个阶段，从活动配置的仲裁处收集信息；在第二个阶段，把信息传播给活动配置的仲裁。请注意，在每个阶段中都可能会发现新的配置。

GeoQuorums是在物理平台上实现原子级共享内存的一种方法。这种物理平台由以任意模式移动的移动节点组成。

为了对其进行处理，每个阶段都通过一个固定点条件终止，该固定点条件包含了各活动配置的仲裁。

让我们再来看下Rambo中的重构，内存访问操作与重构进行解耦，甚至允许在因Recon 使用共识而变慢的情况下终止操作。重构本身包含两种独立的活动：在共识的帮助下，发布一系列一致的配置；然后利用更新进程最终确定重构。在某些场景中，把这两种活动加以整合以满足特殊的性能要求或许能取得明显的优势。例如，RDS服务中的情况。[11]

最后，可以把Rambo 看成一种用于改进和优化的框架，其中的某些内容已经在原型系统中实现（例如，Georgiou [20]等人的著作）。下文描述了对自组织移动网络的Rambo 框架的改造过程。

GeoQuorums[16]是在物理平台上实现原子级共享内存的一种方法。这种物理平台由按任意模式移动的移动节点组成。自组织网络未使用已有的基础设施；与此相反，网络由彼此合作按特定的路由把信息从源传送到目的地的移动节点组成。可以把GeoQuorums 看成具有两层的系统，顶层实现了动态复制的存储系统，底层以通过移动节点实现的静止焦点来提供对象的副本。

焦点为存在地理意义的区域，通常由移动节点“占据”。它可能是一个道路交叉点、一个风景观测点，或沙漠中的水资源。焦点附近的移动节点参与了静止虚对象（称之为焦点对象）的实现。各焦点的实现支持本地广播服务*LBcast*，它提供了可靠的全序广播。LBcast 用于实现一种副本状态机，可容忍移动节点的加入和离开。如果所有移动节点都离开了焦点，那么焦点对象就会失败。

然后，这种方法还确定了用于焦点的仲裁系统集合。每个仲裁系统包括两个集合，分别叫做取值

仲裁（get-quorum）和赋值仲裁（put-quorum），具有每个取值仲裁与每个赋值仲裁相交的特点。使用仲裁后，服务可以容忍一定数量的焦点对象出现失败。为了确保性能，或是处理移动节点的定期迁移，可以安装不同的仲裁系统。

为了便于与焦点对象进行通信，GeoQuorums 假定可以使用 GeoCast 服务（见Imielinski[26]的著作）把消息传送给焦点所在的地理区域。在这种情况下，人们可以使用Rambo 框架作为顶层以实现一个原子内存系统，把焦点作为副本主机。考虑了简洁和效率因素之后，人们对GeoQuorums 方法又做了一些改进。首先处理了重构，然后改进了读写操作的处理方式。

GeoQuorums 首次引入了不依赖共识的通用重构能力。该算法在有限数量的预配置之间进行重构，它使用了与Rambo中升级协议类似的两阶段协议，而不是使用共识。在第一个阶段，调用者联系了与之前所有配置相关的完整取值仲裁（get-quorum）和赋值仲裁（put-quorum）（请注意，即使可能的配置的限定数量很大，最多也只需要获取每个焦点的一对消息）。然后，在第二个阶段中，把信息传给下一个配置的完整赋值仲裁（put-quorum）。

GeoQuorums 改进了用于读写操作的方法，支持某些操作在一个阶段内完成。其实现原理如下：在写的场景中，利用全球定位系统（GPS）的时钟为写入值生成标签，进而确定了写入操作的顺序。采用上述方式后，不再需要像其他实现一样用另一个阶段确定最高的标签。此时的写协议只有一个单一的*put*（*赋值*）阶段与当前配置的任何赋值仲裁（put-quorum）互动。如果写操作监测到了并发的重构，它还会等待从每个配置的赋值仲裁（put-quorum）发出的响应（这时，最多需要与每个焦点联系一次）。一旦写入操作完成，标签就会被确认。在读的场景中，该协议包含一个或两个阶段。首先，*get*（*取值*）阶段为一个典型的查询阶段，从某个完整的取值仲裁获取拥有最大标签的值；如果监测到并发的重构，则该阶段也会等待从各配置的取值仲裁发出的响应（与上文一样，最多需要与每个焦点交换一次信息）。一旦*get*（*取值*）阶段完成后，如果确定可以*确认*获取了的最大标签，读操作终止。在其他情况下，读操作执行*put* 阶段，把最大的标签-值对传给某个赋值仲裁。有关*确认的*标签的信息通过系统进行传播，以支持单阶段的读操作。

动态原子级内存服务在底层分布式平台出现扰动时仍能提供数据一致性和可用性。

使用单独的取值仲裁和赋值仲裁后，可以支持人们对根据读和写的分布情况对系统性能进行调优。如果写入操作比读取操作多，可对系统进行重构以使用更粗的取值仲裁和更细的赋值仲裁，因为写入操作只访问赋值仲裁。如果读取操作更普遍，则系统可以使用更细的取值仲裁和更粗的赋值仲裁，因为某些读取操作只访问取值仲裁。

GeoQuorums 使用实时时间戳（由GPS提供）以加速读取和写入操作，其支持在一个回合内完成写入或某些读取操作。最后，假设存在固定的焦点集合这一条件限制了系统的可演化性，但却能让算法在不使用共识的情况下重构。

从理论到实践

精确一致性的保证、容错以及在动态环境下运行的能力等因素促使研究人员为某些服务构建探索性的实现方式。我们已经论及了Rambo变种的探索性实现。此处，我们又阐述了两种实现。第一种是分布式磁盘阵列的实现[36]，其算法理念源于Rambo。第二种实现[37] 基于DynaStore 算法。

Brick联合阵列（FAB）[36]是惠普实验室开发和评测的存储系统。FAB处理磁盘存储，其中的基本对象为*逻辑块*。该实现的目标是，通过在不干扰客户端请求的情况下分配工作负载，处理故障和实现恢复来提供优于传统主从复制的性能。

FAB 使用了称为*bricks（砖）*的计算机，其中配有商业用磁盘和网卡。为了实现分布式存储，FAB把存储划分为多个逻辑块，并使用纠删码算法把各逻辑块复制到brick的子集中。该系统基于多数仲裁系统。在系统中，为了保证服务的持续时间，在添加或删除bricks时，重构算法会移动数据。客户端通过向一个brick 发送请求来执行读和写操作，在请求中规定了它希望获取或修改的逻辑块。然后，该brick确定存储该块的brick集合，并按照Rambo 算法的规定使用该brick集合运行读或写协议，并使用标签-值对来确定写入值的顺序。写入操作需要两个阶段才能完成，而读操作可能需要一个或两个阶段。如果读取器从所有的brick收到相同的标签，则读操作可以在一个阶段完成。为了提高效率，读取器只向处于空闲状态的活动副本发出请求读取实际的块内容，而对于仲裁中的其他成员，读取器只发出请求读取标签。

对该实现的评测说明，FAB的性能与集中式解决方案的性能类似，但能同时提供不间断的服务和高可用性。

DynaDisk[37]是用于评测的DynaStore 算法实现。[2]研究人员在局域网中测试了该实现。该实现采用了以数据为中心的方法，其中的副本主机为被动网络存储设备。

该实现设计用于在不使用共识的情况下异步地重构服务，或在使用共识的情况下部分同步地重构服务。

该评测说明，在缺少重构的情况下，该算法的两种版本的读写操作延迟相近。根据研究人员的观察，如果多种重构同时发生，那么无共识的异步方法会显著增加与重构并发执行的读写操作的延迟。具体的解释如下：无共识的算法必须检查多种可能的配置，而结合重构与共识的算法通常只要处理所有客户端都认可的一种配置。另一方面，如果多种重构同时发生，那么DynaDisk的无共识版本在重构延迟方面的表现稍微要好一点。在上述场景中，基于共识的DynaDisk 可能需要更长的时间来做出决定。

讨论

我们阐述了在动态分布式系统中实现原子级一致的内存服务的多种方法。在此类系统中，参与节点的集合可能会因节点故障、节点自愿或计划内的离开以及新节点加入计算而随时间发生演进。我们重点关注了原子一致性，因为它是个直观的概念。虽然它包含副本，但是它提供了等同于串行的对象说明的机制。此类服务能让构建分布式应用变得更容易，特别是业界存在下列观点：设计分布式算法时，使用共享内存范式比使用消息传递范式更容易。

本文中阐述的各种方法指出了实现分布式内存服务时可用的不同设计决策。上述任意一种方法都有自己的优缺点。举例来说，基于共识的解决方案虽然在概念上容易理解，但在特定的场景中，它们往往会使用协调器，而其性能则可能极为依赖协调器的可用性。在协调器进行故障切换时，即使多数主机没有发生故障，也有可能会发生严重的延迟。对于具有高链路可用性的低延迟网络，组通信服务是一种有效的构件。然而，视图变化会导致它们承受相当高的开销。即使故障的数量相对较小但蔓延的时间较长，也会造成视图变化。那些在理论上优美的方法（比如使用增量主机替换的DynaStore，使用仲裁替换的Rambo，以及使用焦点的Geo-Quorums）实现起来可能更复杂，但却更灵活，且支持针对特定系统进行优化。能否实现优异的性能还取决于部署平台的稳定性，故障率的假设以及其他因素，比如在FAB定制化实现时考虑的那些因素。

无论如何重构副本主机集合，任何实际的实现都必须应对如何确定*何时* 重构这一挑战。此时，人们可以把决策权委托给独立的主机。举例来说，节点加入服务时会引发重构，或在发现节点故障后，这个节点会隐式地引发重构。在扰动发生频率较低，副本集中成员的数量较小时，此类方法或许简单又有效。但是，如果节点不停的加入和离开服务，即使由主机组成的某个核心集合保持稳定且足以提供良好服务，它们也会造成不必要的开销。另一个方法是把何时重构的决策权留给另一个监测内存系统性能的分布式服务，由它依据监测值和推测的预测值确定何时重构。这种解决方案更加复杂，但是它有可能提供极好的服务质量。还有一种考虑是选择一个合适的主机集合。仅仅因为节点有兴趣作为副本主机并不意味着它的愿望必须得到满足。此时，可确定何时重构的外部服务也能确定节点的目标集合。请注意，无需就目标集合达成一致 —— 所有的内存服务都能处理同时提出几个目标集合的场景。

不管副本主机出现故障的程度和频率如何，动态原子级共享内存服务均可保证所有执行的一致性。然而，读写操作的终止却以受限的故障为条件。在静态系统中，通常很容易做出这种限制：此处，允许任何少数主机的子集出现故障。在动态系统中，对故障模式的限制可能更多地围绕和依赖于特定的算法方法；如果您想了解读取器的其他细节，请阅读本文引用的文献。

出现云服务后，分布式存储服务必然会持续地引起人们的关注。技术上的挑战和性能开销或许是现有分布式存储解决方案回避原子一致性保证的原因。商业解决方案，比如谷歌的文件系统 (GFS),[22] 亚马逊的Dynamo[15] 以及Facebook的Cassandra[28]提供了较不直观且未经证实的保证。在开展生产系统的设计决策时，他们考虑了本次调查所讨论的概念。例如，GFS[22]中使用了共识以确保系统配置的一致，正如Rambo的做法一样；Spanner[13]中使用了全局时间，正如GeoQuorums的做法一样；Dynamo[15]中的副本访问协议使用了仲裁，正如本文调查的某些方法一样。这些例子为研究动态网络系统的一致数据服务注入了动力，促使他们追求严格的算法方法。

一致的存储系统会继续成为活跃研究和高级开发的热点。而且，我们有充分的理由相信，只要具有超级容错能力的高性能动态内存系统问世，它们肯定会在复杂分布式应用的构建中发挥重要作用。对可提供原子读/写内存的实现的要求最终受分布式应用的需求驱动，那些分布式应用需要可证明的一致性和性能保证。

鸣谢

本文工作得到了美国国家自然科学基金会（NSF）的部分资助，资助号1017232。我们在此感谢 Jennifer Welch 和其他匿名的评审员，他们提出了许多见解深刻的评论。

参考资料

1. Aguilera, M., Keidar, I., Martin, J.-P. and Shraer, A. Reconfiguring replicated atomic storage:A tutorial. *Bulletin of the EATCS 102* (Oct. 2010), 84–108.
2. Aguilera, M.K., Keidar, I., Malkhi, D. and Shraer, A. Dynamic atomic storage without consensus.*JACM 58* (Apr. 2011), 7:1–7:32.
3. Attiya, H., Bar-Noy, A. and Dolev, D. Sharing memory robustly in message-passing systems.*JACM 42*, 1 (Jan. 1995), 124–142.
4. Attiya, H. and Welch, J.L.Sequential consistency versus linearizability.*ACM Trans.Comput.Syst.12*, 2 (May 1994), 91–122.
5. Birman, K. A history of the virtual synchrony replication model.Replication:Theory and Practice, *LNCS* vol. 5959 (2010), 91–120.
6. Birman, K., Malkhi, D. and Renesse, R.V. Virtually synchronous methodology for dynamic service replication.Technical report, MSR-TR-2010-151, Microsoft Research, 2010.
7. Brewer, E.A.Towards robust distributed systems, July 2000.
8. Brewer, E.A.Pushing the cap:Strategies for consistency and availability.*IEEE Computer 45*, 2 (2012), 23–29.
9. Calder, B. et al.Windows azure storage:A highly available cloud storage service with strong consistency.In *Proceedings of SOSP '11* (Oct 23-26, 2011), 143–157.
10. Chandra, T.D., Hadzilacos, V. and Toueg, S. The weakest failure detector for solving consensus.*JACM* (1996), 685–722.
11. Chockler, G., Gilbert, S., Gramoli, V., Musial, P.M. and Shvartsman, A.A.Reconfigurable distributed storage for dynamic networks.*J. Parallel and Distributed Computing 69*, 1 (2009), 100–116.
12. Chockler, G., Guerraoui, R., Keidar, I. and Vukolic´, M. Reliable distributed storage.*IEEE Computer*, 2008.
13. Corbett, J.C. et al.Spanner:Google's globally distributed database.In *Proceedings of the 10th USENIX Symp.On Operating Sys.Design and Implementation* (2012), 251–264.
14. De Prisco, R., Fekete, A., Lynch, N.A. and Shvartsman, A.A.A dynamic primary configuration group communication service.In *Proceedings of the 13th Int-l Symposium on Distributed Computing*.Springer-Verlag, 1999, 64–78.
15. DeCandia, G. et al.Dynamo:Amazon's highly available key-value store.In *Proceedings of SIGOPS Oper.Syst. Rev. 41*, 6 (Oct. 2007), 205–220.
16. Dolev, S., Gilbert, S., Lynch, N., Shvartsman, A. and Welch, J. GeoQuorums:Implementing atomic memory in ad hoc networks.In *Proceedings of the 17th International Symposium on Distributed Computing* (2003), 306–320.
17. Dutta, P., Guerraoui, R., Levy, R.R. and Vukolic´, M. Fast access to distributed atomic memory.*SIAM J. Comput.39*, 8 (Dec. 2010), 3752–3783.
18. Fekete, A., Lynch, N. and Shvartsman, A. Specifying and using a partitionable group communication service.*ACM Trans.Comput.Syst.19*, 2 (2001), 171–216.
19. Fischer, M.J., Lynch, N.A. and Paterson, M.S.Impossibility of distributed consensus with one faulty process.*JACM 32*, 2 (1985), 374-382.
20. Georgiou, C., Musial, P.M. and Shvartsman, A.A.Developing a consistent domain-oriented distributed object service.*IEEE Transactions of Parallel and Distributed Systems 20*, 11 (2009), 1567–1585.
21. Georgiou, C., Musial, P.M. and Shvartsman, A.A.Fault-tolerant semifast implementations of atomic read/write registers.*J. Parallel and Distributed Computing 69*, 1 (Jan. 2009), 62–79.
22. Ghemawat, S., Gobioff, H. and Leung, S.-T.The Google File System.In *Proceedings of the 19th ACM Symposium on Operating Systems Principles* (2003), 29–43.
23. Gilbert, S. and Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.*SIGACT News 33* (June 2002), 51–59.
24. Gilbert, S., Lynch, N. and Shvartsman, A. RAMBO:A robust, reconfigurable atomic memory service for dynamic networks.*Distributed Computing 23*, 4, (Dec. 2010), 225–272.
25. Herlihy, M.P. and Wing, J.M.Linearizability:A correctness condition for concurrent objects.*ACM Trans.Programming Languages and Systems 12*, 3 (July 1990), 463–492.
26. Imielin´ski, T. and Navas, J.C. GPS-based geographic addressing, routing, and resource discovery.*Commun. ACM 42*, 4 (Apr. 1999), 86–92.
27. Lakshman, A. and Malik, P. Cassandra:A decentralized structured storage system.*SIGOPS Oper.Syst.Rev. 44*, 2 (Apr. 2010), 35–40.
28. Lamport, L. On interprocess communication.Part I:Basic formalism.*Distributed Computing 2*, 1 (1986), 77–85.
29. Lamport, L. The part-time parliament.*ACM Trans. Comput.Syst.16*, 2 (1998), 133–169.
30. Liskov, B. The power of abstraction.In Proceedings of the 24th Int-l Symposium Distributed Computing. N.A.Lynch and A.A.Shvartsman, Eds.*LNCS*, vol. 6343, Springer, 2010.
31. Loui, M.C. and Abu-Amara, H.H.Memory requirements for agreement among unreliable asynchronous processes.In *Parallel and Distributed Computing, Vol 4 of Advances in Computing Research*.F.P. Preparata, Ed. JAI Press, Greenwich, Conn., 1987, 163–183.
32. Lynch, N. and Shvartsman, A. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts.In *Symposium on Fault-Tolerant Computing*.IEEE, 1997, 272–281.
33. Lynch, N.A.*Distributed Algorithms.* Morgan Kaufmann Publishers Inc., 1996.
34. Martin, J.-P. and Alvisi, L. A framework for dynamic byzantine storage.In *Proc.Intl.Conf. on Dependable Systems and Networks*, 2004.
35. Rodrigues, R., Liskov, B., Chen, K., Liskov, M. and Schultz, D. Automatic reconfiguration for large-scale reliable storage systems.*IEEE Trans. on Dependable and Secure Computing 9*, 2 (2012), 145–158.
36. Saito, Y., Frølund, S., Veitch, A., Merchant, A. and Spence, S. Fab:Building distributed enterprise disk arrays from commodity components.*SIGARCH Comput.Archit.News 32*, 5 (Oct. 2004), 48–58.
37. Shraer, A. Martin, J.-P., Malkhi, D. and Keidar, I. Data-centric reconfiguration with network-attached disks.In *Proceedings of the 4th Int'l Workshop on Large Scale Distributed Systems and Middleware* (2010), ACM, 22–26.
38. Vukolic´, M. Quorum systems:With applications to storage and consensus.*Synthesis Lectures on Distributed Computing Theory 3*, (Jan. 3, 2012).1–146.

Peter Musial（pmmusial@csail.mit.edu）是EMC的首席软件工程师和美国马萨诸塞州剑桥市麻省理工学院计算机科学与人工智能实验室（CSAIL）的准会员。

Nicolas Nicolaou（nicolasn@cs.ucy.ac.cy）是塞浦路斯尼科西亚区塞浦路斯大学计算机科学系的客座讲师。

Alexander A. Shvartsman（aas@cse.uconn.edu）是康涅狄格州斯托斯城康涅狄格大学计算机科学和工程教授。

译文责任编辑：陈海波

DOI:10.1145/2602759

若要查阅相关论文，
请访问 doi.acm.org/10.1145/2602758 rh

技术视角
交互式角色动画的运动场

作者：Michiel van de Panne

计算机图形学长期以来一直在尝试从外观和运动角度来描绘出具有真实感的人物形象。而制作人物动画的一个良好开端是从真实世界中采集人类运动的相关数据。使用预先录制的运动剪辑已经是生成虚拟角色运动的标准组成部分。这些虚拟角色将被大量应用于电影、游戏以及交互式仿真中。当然，现有技术不可能采集到人物可能做出的每个动作，因此动画制作方法试图使用运动采集配置或手工制作动画的方法，尽可能多地处理人们能够录制的有限运动。

有了运动剪辑后，创建新运动的基本思路很简单：播放预先录制的动画剪辑序列，同时注意只有在不造成明显失真的情况下才能切换到新的剪辑。在过渡期间平滑融合两个运动，可进一步帮助增加剪辑间的可行过渡的数量。对于游戏和仿真之类的交互式环境，可以根据用户目标（如所需的特定行走方向或速度）决定接下来要播放哪一个剪辑，这样即可施加控制。一组运动剪辑以及它们之间的可行过渡自然地组成了“运动图”，该图是带有划界过渡点的有向运动图，运动可在过渡点合并或分离。现代游戏引擎发掘了这些概念的各种延伸，如进一步使用融合技术在运动间插值（如具有不同速度和转弯角速度的行走），以及在不失真的情况下对上体和下体使用独立的运动剪辑。但持续的运动仍是从遵循预设路径的一组基础运动中构建的。

然而，上述模型带有一些明显的局限性。首先，如果用户命令突然的转弯或速率改变，运动必须等到下一个可用的剪辑过渡，才能执行命令。目前，很多游戏引擎选择快速剪切或融合到所需的运动，而不是生成因响应时间更长而让玩家感觉迟缓的更高质量的连续运动。其次，运动永远不会真正地偏离预先录制的路径的限制。（至少在没有进一步手动加工运动剪辑融合的情况下不会偏离。）

接下来的论文令人振奋，因为它很大程度上放弃了运动剪辑的思路。取而代之的是，它有效地将运动剪辑处理成一组运动向量。这里运动向量包括角色姿势及其相关速度。整体而言，这些高维运动向量定义了一个运动场，该运动场决定了角色状态如何随着时间而演变。对于任一给定的角色状态，被动动力由平均运动场定义。平均运动场由一组最接近当前状态的运动向量加权平均重构而成。但是，平均运动不容许任何控制，因此会产生逼真但无响应的运动。该论文的关键见解之一在于，它还使用了邻近运动向量来定义一组离散的动作，每个动作表示一种可引导运动未来演变的备选方案。在获得“掌控”角色运动的这种能力后，我们可使用强化学习的方法来很好地针对给定任务目标预计算出最优动作（如以给定的速度沿给定的方向行走）。

接下来的论文令人振奋，因为它将运动剪辑处理成一组独立的运动向量，这里运动向量包括角色姿势及其相关速度。

这种方法的美妙之处在于，合成的运动可随意偏离预先录制的运动剪辑，而同时它们仍能被引导来满足运动目标。实现这种神奇功能的部分原因是：可用的控制操作是从运动场隐式解构的，因此无需显式指定。同样令人振奋的是，该方法支持使用很大程度上非结构化的运动数据，并且概念上易于与其它现有的运动学和动力学方法集成。该论文令人信服地展示了：为了取得新的解决方案而重新思考问题的现有表现形式，能够带来的好处。最后，它展示了一个很有说服力的例子，说明了如何在高维环境中令人信服地应用强化学习的方法。

Michiel van de Panne (van@cs.ubc.ca) 是加拿大温哥华不列颠哥伦比亚大学计算机科学系研究及教务处的教授兼副主任。

译文责任编辑：周昆

DOI:10.1145/2602758

交互式角色运动的运动场

作者：Yongjoon Lee、Kevin Wampler、Gilbert Bernstein、Jovan Popovic 和 Zoran Popovic

摘要

我们提出一种虚拟角色运动数据和运动控制的新型表现形式，这种形式高度灵敏地响应用户输入，并允许自然处理任意外部干扰。与基于直接回放运动数据片段的传统方法相比，我们将运动数据的样本组织成向量场的高维泛化，我们称之为*运动场*。在运行时我们的运动合成机制在运动场中自由流动来响应用户的命令。我们创建的运动看上去很自然，对实时用户输入具有高度响应能力，并且无需在数据中显式指定。

1. 引言

每当视频游戏包含行走或奔跑的角色时，它就需要某种方法来交互式合成这种运动。这种合成远比一开始看起来的要复杂，因为这既要求创建视觉精确的结果，又要求能够交互式*控制*生成哪些运动。实现这种效果的标准技术通过直接（或近乎直接）回放预先录制的动画剪辑来创建逼真的动画。这些技术通过仔细指定什么时候可从播放一段动画剪辑过渡到另一段，来提供控制。因此合成的运动是受限制的，它和预先录制的动画非常接近。

但是，真实的人类运动是一种高度变化且连续的现象：它快速适应不同的任务，并响应外部扰动，并且通常能够从几乎任意初始状态开始连续运动。但是，真实的人类运动是一种高度变化且连续的现象：它快速适应不同的任务，并响应外部扰动，并且通常能够从几乎任意初始状态开始连续运动。遗憾的是，这说起来容易做起来难。例如，尽管角色动画技术有了很多进步，但是创建高度灵敏且逼真的交互式运动控制器仍然是常见但困难的任务。

我们对交互式角色动画提出了一种新的运动表达，我们将它定义为*运动场*，这种运动场提供了两种关键能力：让用户实时控制角色的能力，以及在角色的完全连续的配置空间中进行操作的能力。这就允许了使用*任何*可能的姿势作为动画的有效起始状态，而不是局限于那些跟预录制运动数据相近的姿势。虽然现在已有技术可实现上述的单一能力，但是只有两种能力的组合才可实现高灵敏度的控制器，在短时间里响应用户命令。

更具体地说，运动场是一种映射，它将角色每种可能的配置与一*组*运动相关联，这组运动描述了角色可如何从当前状态变为其他状态。为了生成动画，我们从这一组运动中选择一个运动，根据此运动形成一帧，然后从得到的角色状态开始重复。于是角色的运动会根据这种合成过程在状态空间中“流动”，就像粒子在力场中流动一样。但是与单一固定流不同，运动场允许在每帧有多种可能运动。通过使用强化学习方法，在运行时智能选择这些可能的运动，流的方向就可以更改，从而允许角色以最优方式响应用户命令。

由于运动场允许每个帧有一系列动作，因此角色可立即响应新的用户命令，而不是像在运动图中一样，等待不同动画剪辑之间预先确定的转换点。这就使得基于运动场的控制器比基于运动图的控制器灵敏得多。通过其它外部方法进一步改动此流，如逆向运动学或物理仿真，我们还可以直接将这些技术集成到运动合成和控制过程中。进一步的，因为我们的方法只需要运动采集数据中非常少的结构，所以生成新控制器的工作量是最小化的。

2.相关研究

过去十年里，交互式角色动画的主要方法一直是基于小心回放预先摄制的动画剪辑。通过检测在哪里可以将一段动画剪辑过渡到另一端（没有出现明显的视觉“跳跃”），这些方法可实现真实感运动以及实时控制。为了更好地理解后面阐述的我们的运动场方法，从几何的角度来展示这几类方法的工作原理会很有用。角色肢体所有可能姿势组成的完整空间有数十个维度，但我们可以从二维（2D）平面来抽象地展示它。平面中的每个点表示角色的一个静态姿势，而动画就通过一段连续路径来表示。如果我们将此 2D 平面想象成类似大学庭院的某种东西，那么我们就可以沿着走过庭院时形成的路径“勾绘出”动画。

本文的最初版本发表在 *Proceedings of the ACM SIGGRAPH Asia 2010*。

对于直接依赖于回放预先摄制的动画剪辑的方法，[1, 6, 7]我们可以想象这些录制的剪辑形成了庭院中铺设的很多路径。每个路径表示动画数据，路径相遇或分叉的点对应于运动可从一个动画平滑过渡到另一动画的点。直观地说，这些基于剪辑的方法对应了这样的限制：即一个人总是在铺设的路径上行走，这就代表了那些运动可被合成为图的方式。这种限制尽管意味着所有合成的运动都和数据密切吻合，但是大部分可能运动的空间都成了禁区。这种情况所带来的缺点是，运动难以快速响应用户命令的变化或意外扰动，因为只有在到达图中新边的时候，运动才能发生变化。[11, 16] 其次，由于运动被限制在了那些组成图的剪辑范围内，因此很难将这些方法耦合到物理模拟器以及其它可扰动运动状态使之偏离图中状态的技术。更一般的情况是，当角色想从任意状态配置开始运动时，我们很难使用上述基于图的控制器。[20] 本质上，我们的运动场方法通过允许动画“偏离路径”来解决这些问题，并且此方法仅将一组预先摄制的动画用作粗略的指南来确定要合成哪个运动。

虽然已有很多方法被提出来缓解单纯的基于图的控制器带来的表达上的弱点，包括参数化运动图、[5, 14]增加可能的过渡数量，[2, 19]以及在图结构中拼接玩偶系统的动力，[20]但是根本问题仍未解决：除非表现形式以一种实时可控的方式规定每一个连续状态的运动，否则角色的运动将仍然受限。因此，即使此类方法提前预料到某些用户输入，[11]角色仍然可能反应过慢，或者过渡得过于唐突，因为图中没有更短的路径。同样，当方法预测到某些类型的上半身推动时，[2]角色可能对手的拉动或下半身推动毫无反应。

另一组方法使用非参数模型来学习完全连续空间中的角色运动的动力情况。[3, 17, 18]这些技术通常能够从任意初始状态开始合成运动，并且能很好地施加物理扰动[18]以及根据非完整数据估计角色姿势。[3]这些模型用于估计角色在每一种可能的状态下执行的单个“最可能”的运动。这就丢失了最优控制角色的能力。我们工作和这些工作的主要区别在于，我们没有为最可能发生的单一运动构建模型，而是尝试在每个角色状态建模出一组可能的运动，然后在运行期间使用最优控制理论的原理选择单个运动。这使得我们能在享受完全连续状态空间所带来的好处的同时，交互式地控制角色。我们的工作结合了基于图的方法中的近似最优角色控制的概念以及非参数运动估计技术的概念。

尽管我们的控制器属于运动学范畴（因为除了通过构造控制器的数据，控制器不会直接引入物理内容），但是动态控制器作为角色动画的另一种方法已经得到广泛研究。原则上，这样的控制器最有可能实现高度逼真的交互式角色动画。但是，高保真、基于物理的角色动画更难实现，因为单靠物理学并未告诉我们推动角色所需的肌肉力，并且灵敏、逼真、完全动态的角色设计仍是未解决的难题。

3.运动场

视频游戏等交互式应用需要角色能够快速响应用户的命令和意外的干扰，同时还要保持生成动画的可信度。理想的方法是对自然人类运动的完整空间进行完全建模，从而描述了角色从给定状态可运动的每种可能方式。此类模型可使得运动在连续运动空间中有更高的灵活度和灵敏度，而不是将运动限制在预先录制的运动剪辑和过渡上。

虽然不太可能对自然角色运动的整个空间进行完全建模，但是我们可以将运动捕获数据用作局部近似。我们提出了一种称为*运动场*的结构，这种结构查找并使用与角色在任意点的当前运动相似的运动捕获数据。通过参考相似运动来确定哪些未来行为合理，我们就可确保我们的合成动画保持自然：与运动捕获数据相似，但是极少相同。这将角色从简单的回放运动数据中解放出来，使其能够在数据的整体邻近区域自由移动。而且，因为总是有多种要参考的运动数据，因此角色始终有各种方法来进行运动中的快速变化。

本节中，我们将介绍如何基于一组示例动画剪辑来定义运动场。我们将暂时忽略运动场如何用于交互式控制角色的问题，而是将重点放在合成非交互动画上。为此，我们将一组示例动画剪辑组织成运动场来描述单个“流”— 从而允许从任何可能的角色姿势开始生成动画。在第 4 节中，我们将介绍如何扩展这种技术来使角色得到实时交互式控制。

3.1.初步定义

运动状态。 在我们将介绍的状态下，角色可由姿势以及角色所有关节的速度来配置。*姿势* $x = (x_{\text{root}}, p_0, p_1, \ldots, p_n)$ 由 3D 根位置向量 x_{root}、根朝向四元数 p_0 以及关节旋转四元数 $p_1, \ldots, p_n$ 组成。根点位于骨盆。*速度* $v = (v_{\text{root}}, q_0, q_1, \ldots, q_n)$ 由 3D 根偏移向量 v_{root}、根偏移四元数 q_0 和关节偏移四元数 $q_1, \ldots, q_n$ 组成 — 所有这些量通过有限差分找到。给定两个姿势 x 和 x'，我们可以计算得到以下有限差分

$$v = x' \ominus x = (x'_{\text{root}} - x_{\text{root}}, p'_0 p_0^{-1}, p'_1 p_1^{-1}, \ldots, p'_n p^{-1}{}_n)$$

通过倒置上述差分，我们可以将速度 v 添加到姿势 x，以获得新的偏移后的姿势 $x' = x \oplus v$。我们还可以在姿

势或速度的对应部件上使用向量线性插值或单位四元数插值[13]，将多个姿势或速度一起插值（$\sum_{i=1}^{k} w_i x_i$ 或 $\sum_{i=1}^{k} w_i v_i$）。我们使用 $\ominus$、$\oplus$ 和 $\sum$ 来类比笛卡尔空间中的向量相加和相减，但用圆圈来提醒读者我们主要在处理四元数。

最后，我们将*运动状态* $m = (x, v)$ 定义为姿势和关联的速度，根据一对连续的姿势 x 和 x' 用 $m = (x, v) = (x, x' \ominus x)$ 来计算。所有可能的运动状态的集合构成高维连续空间，其中每个点表示角色在单个瞬间的状态。通过此空间的路径或轨迹表示角色的连续运动。在讨论动力系统时，此空间通常称为相空间。但是，由于我们的运动合成属于运动学范畴，因此我们改用*运动空间*这一词组以避免混淆。

运动数据库。我们的方法将一组运动捕获数据作为输入，然后构造一组运动状态 $\{m_i\}^n{}_{i=1}$，术语称为*运动数据库*。此数据库中的每个状态 m_i 根据一对连续帧 x_i 和 x_{i+1}，通过上述方法 $m_i = (x_i, v_i) = (x_i, x_{i+1} \ominus x_i)$ 来构造。我们还计算并存储下一对帧的速度，用 $y_i = x_{i+2} \ominus x_{i+1}$ 来计算。通常，来自数据库的运动状态、姿势和速度将加注下标（如 m_i、x_i、v_i 和 y_i），而任意状态、姿势和速度无下标出现。

相似性和邻域。我们的运动场定义的核心是运动状态间的*相似性*这一概念。给定运动状态 m，我们通过对数据库的 k 近邻查询，计算出 k 个最相似运动状态的*邻域* $\mathcal{N}(m) = \{m_i\}^n{}_{i=1}$。[12]我们的测试使用 $k = 15$。我们用

$$d(m, m') = \sqrt{\begin{aligned} &\beta_{\text{root}} \left\| v_{\text{root}} - v'_{\text{root}} \right\|^2 + \\ &\beta_0 \left\| q_0(\hat{u}) - q'_0(\hat{u}) \right\|^2 + \\ &\sum_{i=1}^{n} \beta_i \left\| p_i(\hat{u}) - p'_i(\hat{u}) \right\|^2 + \\ &\sum_{i=1}^{n} \beta_i \left\| (q_i p_i)(\hat{u}) - (q'_i p'_i)(\hat{u}) \right\|^2 + \end{aligned}} \tag{1}$$

来计算相似（相异）性，其中 $\hat{u}$ 是某种任意单位长度向量；$p(\hat{u})$ 表示 $\hat{u}$ 旋转 p；权重 β_{root}、β_0、β_1、…、β_n 是可调标量参数。在我们的试验中，我们将 β_i 设为身体在关节 i 处的骨骼长度（单位米），β_{root} 和 β_0 设置为 0.5。直观来看，将 β_i 设置为其关联骨骼的长度将弱化手指等小骨骼的影响。请注意，我们将分解出根世界位置 (root world position) 和根偏航方向 (root yaw orientation)（但不是其相应的速度）。

相似性权重。因为我们允许角色脱离数据库中的运动状态，因此我们不得不经常从邻域 $\mathcal{N}(m)$ 中插值数据。我们将用于此类插值的权重 $[w_0, \ldots, w_k]$ 称为*相似性权重*，因为它们测量与当前状态 m 的相似性：

$$w_i = \frac{1}{\eta} \frac{1}{d(m, m_i)^2} \tag{2}$$

其中 m_i 是 m 的第 i 个近邻，$\eta = \sum_i \frac{1}{d(m, m_i)^2}$ 是确保权重总和为 1 的归一化因数。

3.2.运动合成

动作。处于运动状态 m 的*运动场* $\mathcal{A}$ 的值是控制动作 $\mathcal{A}(m)$ 的集合，这确定了角色在单帧的时间里可过渡到哪些状态。这些操作中的每个操作 $a \in \mathcal{A}(m)$ 指定了近邻 $a = [a_1, \ldots, a_k]$（$\sum a_i = 1$ 且 $a_i > 0$）的凸组合。将某些 a_i 权重的值增大到超过其他权重值后，动作的方向将偏离，以更好地匹配关联近邻的方向（图 1）。给定一个特定操作 $a \in \mathcal{A}(m)$，然后我们使用过渡或*积分函数* $m' = (x', v') = \mathcal{I}(x, v, a) = \mathcal{I}(m, a)$ 来确定下一个状态 m'。设 i 的范围遍及邻域 $\mathcal{N}(m)$，我们使用函数

$$\mathcal{I}(m, a) = \left(x \oplus \sum a_i v_i, \sum a_i y_i \right) \tag{3}$$

遗憾的是，此函数经常造成角色的状态漂移到某些区域，在这些区域中，我们几乎没有掌握关于角色应如何移动的数据，从而导致不逼真的运动。为了纠正此问题，我们使用一个很小的漂移校正项，它不断将我们的角色拉向数据库中最接近的已知运动状态 $\bar{m} = (\bar{x}, \bar{v})$。这种拉拽的强度由参数 $\delta = 0.1$ 来控制：

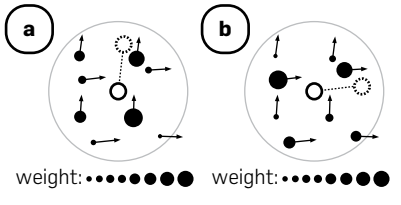
$$\mathcal{I}(m, a) = (x \oplus v', y') \tag{4}$$

$$v' = (1 - \delta) \left(\sum_{i=1}^{k} a_i v_i \right) \oplus \delta \left((\bar{x} \oplus \bar{v}) \ominus x \right) \tag{5}$$

$$y' = (1 - \delta) \left(\sum_{i=1}^{k} a_i y_i \right) \oplus \delta \bar{y} \tag{6}$$

被动动作选择面对可选择的各种动作，现在我们知道了如何生成动画，但是应该选取哪一个动作呢？这个问题主要是第 4 节的主题。但是，我们可以使用相似性权重（等式 (2)）作为我们选择的动作来快速实

图 1.*使用动作权重进行控制*。通过为当前状态（白点）的近邻（黑点）重新赋予权重，我们可以控制运动合成来将我们的角色引向不同的下一状态（虚线点）。(a) 产生向上运动的权重。(b) 产生向右运动的权重。



现一个简单的解决方案。此选择会造成角色蜿蜒地穿过数据，形成逼真（但漫无目标）的人类运动流。

4.控制

如第 3 节所述，在角色每一个可能的状态上，运动场提供了一*组*动作，角色可从中选择动作来确定其下一帧的运动。通常，应该从此集中选择哪个特定动作，这取决于用户的当前命令。因此，实现实时交互式运动控制器的关键就在于，在每个状态下确定要选择哪个动作来响应用户命令。

4.1.Markov 决策过程控制

从用户的角度看，控制虚拟角色最简便的方法是通过高级命令，如“向右转”或“向后走”。最终，这些高级命令必须转化为执行它们所需的低级操作。就运动场而言，这些低级操作是在每一帧选定的。因为这些低级选择发生在很短的时间内，因此应该作何选择来满足用户高级指令并不是显而易见的。因此，为便于用户控制，有必要将在运动场中使用的单帧操作与其长期结果关联起来。

有效规划短期操作的长期结果是人工智能领域的标准问题。特别要指出的是，称为*强化学习*的人工智能领域提供了相应的工具，可以自然地应用这些工具来通过运动场控制角色动画。为了应用这些强化学习工具，我们将运动场控制问题表达为 Markov 决策过程 (MDP)。

MDP是一种数学结构，公式化地表达了根据当前结果和长期结果来做决策的概念。MDP 由四个部分组成：(1) 状态空间，(2) 每个状态下要执行的动作，(3) 确定由动作产生的状态过渡的方法，以及 (4) 对占据所需状态和执行所需动作的报酬。通过按此框架表达角色动画任务，我们可使角色意识到其动作的长期结果。这在基于图形的控制器中也很有用，但是对运动场控制器却更为关键，因为我们将操作每一帧，而不是每一个剪辑。有关基于 MDP 的控制的更多背景信息，请参阅 Sutton 和 Barto[15] 或者 Treuille[16] 以及 Lo 和 Zwicker[10]，了解它们在基于图的运动控制器中的使用。

状态。将角色状态简单表示为运动状态 m 对于交互式控制来说并不充分，因为我们还必须表示角色在多大程度上完成用户指定的任务。因此，我们增加*任务参数* θ_T 的向量来跟踪任务执行的良好程度，形成关节*任务状态* $s=(m,\theta_T)$。例如，我们的方向跟踪任务 θ_T 记录了一个数值 — 偏离所需方向的角度。用户通过更改此值来控制角色的方向。

动作。在每个任务状态 $s=(m,\theta_T)$ 上，运动场中的角色有一组动作 $\mathcal{A}(m)$，可以从中选择动作来决定角色如何过渡到下一帧（第 3 节）。在 $\mathcal{A}(m)$ 中有无穷多的不同动作，但是用来解决 MDP 的很多技术需要每个状态有*有限*数目的动作。为了满足我们的 MDP 控制器的这一要求，我们从 $\mathcal{A}(m)$ 中采样有限数目的动作 $\mathcal{A}(s)$。给定运动状态 m，我们通过修改相似性权重（等式 (2)）来生成 k 个动作。每个动作旨在从众多邻居中优先选择一个邻居。

$$\left\{\frac{a_i}{\|a_i\|}\,\middle|\,a_i=(w_0,\ldots,w_{i-1},1,w_{i+1},\ldots,w_{k-1})\right\} \tag{7}$$

换言之，为了得到动作 a_i，只需将 w_i 设置为 1 并重新归一化。此方案采样了与 m 的被动动作差别不大的动作，以避免运动不流畅，同时让角色获得足够的灵活性来前移至附近的运动状态。

过渡。此外，我们必须扩展积分函数 $\mathcal{I}$（等式 (4)）的定义来处理任务参数：$\mathcal{I}_s(s,a)=\mathcal{I}_s(m,\theta_T,a)=(\mathcal{I}(m,a),\theta'_T)$.如何更新任务参数通常非常明显。例如，在方向跟踪任务中，θ_T 是角色偏离所需方向的偏移角，所以我们只需按角色转弯的角度来调整 θ_T 即可。

报酬。为了使我们的角色执行所要的任务，我们提供了报酬。正式而言，*报酬函数*指定一个实数 $R(s,a)$，量化在状态 s 下执行动作 a 而获得的报酬。例如，在我们的方向跟踪任务中，对于偏离所需方向较小的偏移，我们给予较高的报酬 $R(s,a)$，而对较大的偏移则给予较低的奖励。具体任务参数以及演示中所用的报酬函数，请见第 5 节。

4.2.强化学习

强化学习的目的是找到“最佳”规则或*策略*来选择要在任意给定状态执行的动作。解决此问题的一个简单方法是选择产生最大当前报酬的动作— 即*贪婪策略*：

$$\pi_G(s)=\operatorname*{argmax}_{a\in\mathcal{A}(s)}R(s,a) \tag{8}$$

此策略虽然简单，但是短视，它忽略了每种动作选择的未来后果。我们已经知道，基于图的贪婪控制器表现很差。[16] 运动场甚至更糟。就算是对于简单的改变方向任务，我们也需要远远超过 1/30 秒的时间范围来预测和执行转向。

我们需要以某种方式考虑当前动作选择对角色获得未来报酬的能力所产生的影响。*前瞻策略* π_L 正好考虑到了这一点，它考虑未来任务状态的累积报酬：

$$\pi_L(s)=\operatorname*{argmax}_{a\in\mathcal{A}(m)}\left[R(s,a)+\max_{\{a_t\}}\sum_{t=1}^{\infty}\gamma^t R(s_t,a_t)\right] \tag{9}$$

其中 $s_1=\mathcal{I}_s(s,a)$，$s_t=\mathcal{I}_s(s_{t-1},a_{t-1})$。$\gamma$称为*折扣因子*，它控制角色相对于长期 ($\gamma\to 1$) 报酬，在多大程度上注重短期 ($\gamma\to 0$) 报酬。

如前所述，计算前瞻策略不仅涉及求解最优下一动作，还涉及求解最优未来动作的*无限*序列。尽管这看起来不切实际，但有一个标准技巧却能让我们有效解出正确的下一个动作。此技巧先定义一个*值函数* $V(s)$，这是一个标量值函数，表示从任务状态 s 开始的最佳动作应该得到的累积未来报酬：

$$V(s) = \max_{a \in \mathcal{A}(m)} \sum_{t=0}^{\infty} \gamma R(s_t, a_t) \quad (10)$$

我们将简短描述我们如何表示和预计算该值函数，但目前请注意，我们现在可以重写等式 (9)，将无限未来搜索替换为值函数查找：

$$\pi_L(s) = \operatorname*{argmax}_{a \in \mathcal{A}(m)} \left[R(s,a) + V(\mathcal{I}_s(s,a)) \right] \quad (11)$$

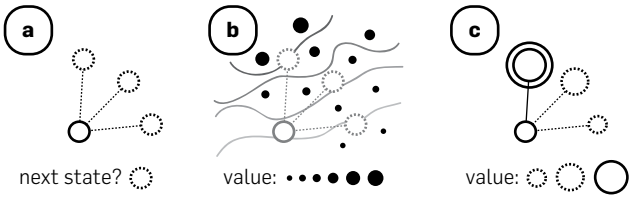
现在计算前瞻策略的开销只比计算贪婪策略高一点。

值函数表示与学习。由于存在无限多可能的任务状态，因此我们无法确切表示值函数。但是，我们可以近似估计此函数，方法是存储有限数量任务状态 s_i 的值并进行插值来估计其他点的值（图 2）。我们通过取数据库运动状态 m_i 和问题的任务参数上的统一网格采样这两者的笛卡尔积来选择这些任务状态样本（请参阅第 5 节了解采样的详情）。此采样给了我们在运动数据库状态附近的高分辨率结果，这正是处于角色通常保持的状态。为了计算不在数据库中的任务状态的值 $V(s)$，我们使用相似性权重在邻近运动状态间插值，并在任务参数间进行多线性插值。

给定从运动场导出的MDP和任务指定，我们使用*拟合值迭代*求出此形式的近似值函数。[4]拟合值迭代在运算时首先表明等式 (11) 可用于以递归形式编写值函数的定义。我们用其他任务状态样本的值来递归表示任务状态样本 s_i 的值：

$$V(s_i) = R(s_i, \pi_L(s_i)) + V(\mathcal{I}_s(s_i, \pi_L(s_i))) \quad (12)$$

图 2.*使用值函数的动作搜索*。(a) 在每个状态，我们有多种可能的动作（虚线）及其下一状态（虚线圈）。(b) 我们插值存储在数据库状态（黑点）的值函数，以确定每一个下一状态的值。(c) 我们选择最高值动作来执行。



其中 $\pi_L(s_i)$ 如等式 (11) 中的定义，$V(\mathcal{I}_s(s_i, a))$ 是通过插值来计算的。通过反复套用等式 (11) 和 (12)，我们可以解出每个样本状态的 $V(s_i)$。我们先从每个样本 s_i 的全零值函数 $V_0(s_i) = 0$ 开始。然后，对每个 s_i 使用等式 (11) 计算 $\pi_L(s)$，之后我们使用等式 (12) 来确定 s_i 的更新值。这样处理完所有 s_i 样本后，我们得到更新后的值函数近似。我们重复此过程直至收敛，然后将最后的迭代用作最终值函数。

时间值函数压缩。与基于图的方法不同，运动场可使角色在很多数据源之间不断过渡。因此，我们需要获得所有运动状态的值函数，而不是只获得剪辑间过渡的值函数。相对于图，这种情况会导致很大的内存占用量。我们通过压缩来弥补这一弱点。

为了实现压缩，我们将我们的值函数视为一个值函数集合，每个值函数在任务参数的空间上定义。若不压缩，我们会在每个数据库运动状态 m_i 上存储其中一个值子函数 $V_{mi}(\theta_T) = V(m_i, \theta_T)$（请见图 3）。在这里，我们观察到，我们的运动状态最初是从连续的运动数据流获取的。在 30 Hz 时，时间上相邻的运动状态及其值函数经常相似；我们期望 V_{mt} 相对于原始剪辑时间会随着“连续”运动状态 m_t 平滑变化。利用这个概念，我们仅仅每隔 N 个运动状态存储一个值函数，并差值得到其它数据库运动状态的值函数（请见图 4）。我们将这些存储值函数的数据库状态称为“锚点”运动状态。我们将两个锚点 m_0 和 m_N 之间第 i 个运动状态的值函数计算为

图 3.*未压缩值函数*。值函数 V_{mi} 存储在每一个运动状态 m_i 中。

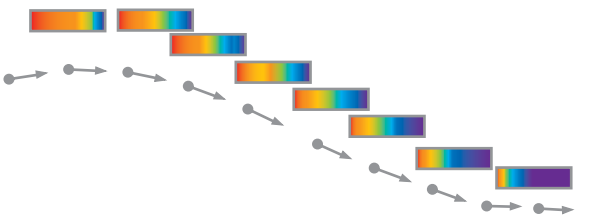
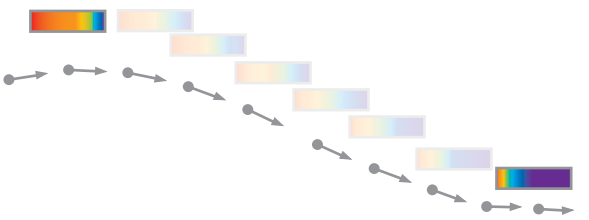


图 4.*时间上压缩的值函数*。中间运动状态的值函数由已显式存储值函数的相邻“锚点”运动状态插值。



$$V_{m_i}(\theta_T) = \frac{N-i}{N} V_{m_0}(\theta_T) + \frac{i}{N} V_{m_N}(\theta_T) \quad (13)$$

我们可以得到一个时间上压缩的值函数，它是第 4.2.1 节中提供的算法的轻微改动形式。我们只迭代与锚点运动状态关联的那些状态，而不是迭代所有任务状态。

此技术容许在基于运动场的控制器的灵敏度与其内存需求之间进行权衡取舍。执行很少时间插值或不执行时间插值会产生非常敏捷的控制器，但代价是内存更多；而采用大幅度时间压缩的控制器往往灵敏度较低。我们在实验中发现，采用时间压缩的运动场控制器在被限制为使用等量的内存时，与基于图形的控制器差不多一样灵敏，而使用适度增加的内存，敏捷度大大提高（请见第 5 节）。

5.实验

本节分析运动场的两个重要属性 — 响应用户指令更改的灵敏度，以及响应动态微扰的能力。

5.1.对用户控制的灵敏响应

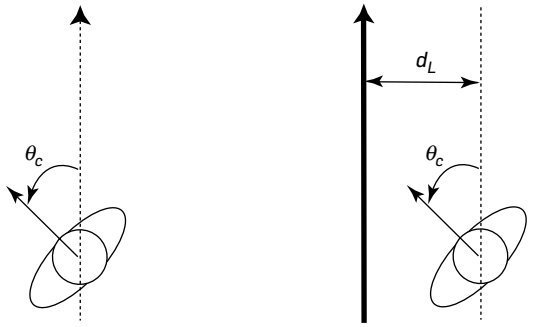
实验设置。我们为两个示例任务创建了值函数：跟踪用户指定的任意方向，以及在跟踪用户方向的同时保持直线（请见图 5）。方向任务的报酬 $R_{\text{direction}}$ 和线路跟踪任务的报酬 R_{line} 分别定义为

$$R_{\text{direction}}(m, \theta_c, a) = -|\theta_c| \quad (14)$$

$$R_{\text{line}}(m, \theta_c, d_L, a) = -|\theta_c| - 0.05|d_L| \quad (15)$$

运动数据设置。 我们使用 142 秒的运动数据，其中包含步调悠闲的移动和对方向和线路变化的快速响应。我们尽量随意地选择源运动数据，但确保大致覆盖可能存在运动的空间。唯一的手动预处理是足部接触标注。

图 5. *任务参数*。对于方向跟踪任务 (a)，使用了预期方向和角色面对的方向的角度差 θ_c。对于线路跟踪任务 (b)，还要考虑预期线路的距离 d_L 以及 θ_c。



(a) Direction　　(b) Line Following

值函数计算。 我们使用值迭代来计算值函数。对于方向任务，我们存储 18 个统一取样方向 θ_c 的值。对于线路跟踪任务，我们采用 18 个统一 θ_c 样本与 13 个统一 d_L 样本（跨度为 −2.0 到 2.0 米）之间的笛卡尔叉积取样。我们将折扣因子设置为 $\gamma = 0.99$。对于每个任务，我们还创建了值函数的“时间压缩”版本，在这些版本中，我们在等式 (13) 里设置 $N = 1, 10, 20, 30$。如果有足够的内存来缓存动作和过渡，则使用值迭代来求解值函数只需要不到 2 分钟，否则需要 3 小时。将这些值迭代更新运算分摊到计算机集群，即可轻松克服这些时间和内存负担。

响应时间分析。

基于图的控制 vs 运动场控制。 为了比较角色能多快适应突然变化的指令，我们使用相同的运动数据、任务和报酬函数来创建基于图的任务控制器[8]。为了最大程度提高灵敏度，我们容许剪辑上有一系列最大 ±45 度的方向偏离，并赋予身体开销最低重要度（详情请见 Lee 等人[8]）。图 6 显示了对不断变化的用户方向的典型响应。对于这两种任务，运动场都证明了能更快地收敛到新目标（如随附的视频和表 1 所示）。

值函数压缩的影响。 我们对统一采样的用户方向变化

图 6. *响应时间*。随时间而变的方向调整，4.23 秒内三次连续方向更改。运动场控制调整的时间远远短于基于图的控制。

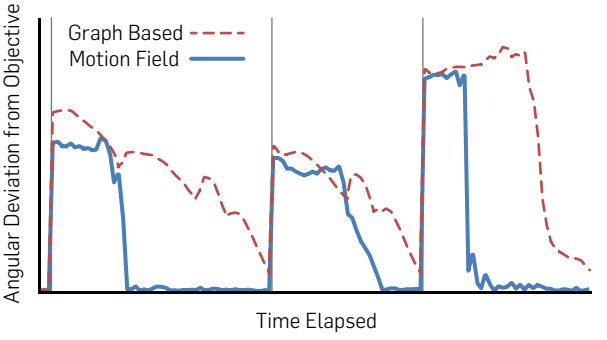


表 1. 方向任务响应时间（单位秒），即收敛到预期方向 5 度以内所用的时间

表现形式	最小	平均	最大
基于图	0.31	0.94	2.36
运动场	0.21	0.40	1.01
运动场 ×10	0.21	0.49	1.23
运动场 ×20	0.25	0.66	1.19
运动场 ×30	0.38	0.78	1.93

运动场 ×10 表示运动场的时间十倍压缩的值函数 (N = 10)。运动场 ×20 和 ×30 定义相似。时间三十倍压缩的运动场的灵敏度与基于图形的控制相似，而即使是二十倍的压缩，其响应也远远快于基于图形的控制。

使用压缩的值函数来记录响应时间。随着压缩度不断增加，系统仍然可靠地实现了目标，但是渐渐失去了最初响应的灵敏度（请见表 1）。我们对线路跟踪任务运行了相似的实验。我们对用户方向变化和线路位移变化进行了统一采样。然后，我们测量时间，直至角色收敛到偏离所要的方向 5 度且距离所要的跟踪线 0.1 米的范围内。我们观察到灵敏度损失是相似的（请见表 2）。

存储要求和计算负载。方向跟踪任务的未压缩值函数存储了 320KB。对于 10 倍、20 倍和 30 倍的案例，压缩值函数分别需要 35KB、19KB 和 13KB。这与基于图的方法所需的存储量 (14KB) 相当。我们认为这很合理，并且实现了存储与灵敏度之间的灵活取舍。对于更复杂的任务，这些值函数大小的增长与基于图的值函数的大小增长相一致。

近似最近邻域 (ANN)[12] 查询占据了大部分计算开销。运行时性能取决于样本动作规模 k（等式 (7)），因为我们发出了 $(k+1)$ 次 ANN 调用来查找最优动作：一次 ANN 调用查找当前状态的近邻，然后另外 k 次调用查找下一状态的近邻，以便通过插值求解。我们认为，局部化的邻域搜索可减少 n 次后续调用的开销，因为后面的状态在 30 Hz 时往往相互之间非常接近。

同样的 ANN 开销在学习时也适用。对于大型数据库或高维任务，简单的学习实现方案需要数小时才能学习一个值函数。通过缓存对固定运动样本的 ANN 调用的结果，我们可以将学习时间大大缩短到只要几分钟。

5.2.微扰

由于每个运动状态由一个姿势和一个速度组成，因此角色可占据的运动状态空间与角色的相空间（被视作动态系统）完全相同。这种相同性使得我们可以轻松应用任意物理或非物理微扰和调整。例如，我们可以纳入动力引擎或反向运动学。而且，为了定义一个恢复的运动，我们不必依赖于目标姿势或轨迹跟踪。作为我们运动合成和控制算法的副产品，恢复可以自动且与扰动同时发生。

表 2.线路跟踪任务响应时间（单位秒），即收敛到所需方向 5 度以内，且距离预期的跟踪线 0.1 米以内的时间

表现形式	最小	平均	最大
基于图	0.47	1.30	2.19
运动场	0.30	0.57	1.26
运动场 ×10	0.30	0.68	1.42
运动场 ×20	0.42	0.91	2.51
运动场 ×30	0.55	1.45	3.56

在此二维控制示例中，二十倍压缩仍比基于图的控制响应快。

为了测试我们的合成算法中融入的微扰，我们将伪物理交互纳入运动场驱动的合成。这是使用物理模拟器完成的，我们使用物理模拟器来确定角色如何响应推或拉，并将此模拟的结果融入运动场流。我们对以下四个数据集使用被动和受控运动场测试了微扰：

1. 18 次行走，包括斜向、后退和蹲伏；
2. 数据集 1 外加 7 次走着推动和 7 次站着推动；
3. 5 次行走、6 次站着手臂拉动、6 次行走中手臂拉动、7 次站着躯干推送，和 7 次行走中躯干推送；
4. 14 次行走和转向。

角色逼真地响应小到中度扰动，即使在只包含非推动运动捕捉的数据集 1 和 4 中也是如此。在包含推动数据的数据集 2 和 3 中，我们观察到更多样的逼真响应以及对更大扰动的更出色处理。施加于角色身体不同部位的力通常使角色产生相应的反应，即使在存在用户控制的情况下也是如此。

但是，我们观察到在某些情况下，作用力产生了非逼真的运动。当角色被推入的状态远离包含合理响应的数据时，就会发生这种情况。只需包含更多推动运动的数据即可解决此问题。

6.讨论

本文介绍了角色运动和控制的新表现形式，此形式允许实时受控的运动在角色姿势的连续配置空间中流动。此流可响应用户提供的实时任务而改变。因其连续的性质，它解决了类似图的表现形式的离散本质所固有的某些关键问题，包括灵敏度和响应度、从任意姿势开始的能力，以及对微扰的响应。而且，这种表现形式无需预处理数据，也无需确定在何处连接所捕获数据的剪辑。这使得我们的方法灵活、易于实现且易于使用。我们认为，无结构技术（比如我们提出的技术）将提供一种有用的工具来实现高度响应的交互式角色，这样的角色是创建可信的虚拟角色所必需的。

虽然运动场表现形式可以单独使用，但我们认为它可轻松与基于图的方法相结合。由于运动场对其底层数据要求极少，因此它们可直接增强基于图的表现形式。这样，当运动可以被安全地限制在图中时，我们可以获得图的好处（计算效率、易于分析等），而当运动离开了图（如因为扰动）或需要极端响应能力时，我们仍有能力来处理。

与任何其他数据驱动的方法一样，我们的方法也受限于它所获得的数据。只要角色一直接近数据，合成运动就会看上去非常真实。当角色远离数据时，运动的真实性和物理合理性就会下降。虽然总是受到数据存在与否的限制，但是较新的研究即使在模型是使用相对少量

的运动数据构造的情况下，使用高斯过程潜变量模型仍取得了不错的表现，[9]我们还希望将物理动力学概念（惯性、重力等）引入到合成过程来拓展合理运动的范围。

更宽泛地说，我们认为运动场为运动表现形式提供了有用的起点，从这里可以突破严格结构化的状态概念。我们相信，无结构运动技术（如我们的技术）具有大幅提高虚拟角色逼真度和响应度的潜力，它们对动画问题的适用性将随着人们开发出更好的距离度量、集成技术以及更有效的搜索和表现方法而不断提高。

鸣谢

本研究得到 UW Animation Research Labs、Weil Family Endowed Graduate Fellowship、UW Center for Game Science、Microsoft、Intel、Adobe 和 Pixar 的支持。

参考资料

1. Arikan, O., Forsyth, D.A.Interactive motion generation from examples. *ACM Trans.Graph.(ACM SIGGRAPH 2002) 21*, 3 (2002), 483–490.
2. Arikan, O., Forsyth, D.A., O' Brien, J.F. Pushing people around.In *Proceedings of the 2005 ACM SIGGRAPH/ Eurographics Symposium on Computer Animation* (2005), 59–66.
3. Chai, J., Hodgins, J.K.Performance animation from low-dimensional control signals.*ACM Trans. Graph.24* (2005), 686–696.
4. Ernst, D., Geurts, P., Wehenkel, L., Littman, L. Tree-based batch mode reinforcement learning.*J. Mach.Learn. Res.6* (2005), 503–556.
5. Heck, R., Gleicher, M. Parametric motion graphs.In *Proceedings of Symposium on Interactive 3D Graphics and Games (I3D) 2007* (Apr. 2007).
6. Kovar, L., Gleicher, M., Pighin, F. Motion graphs.*ACM Trans.Graph.21*, 3 (Jul.2002), 473–482.
7. Lee, J., Chai, J., Reitsma, P.S.A., Hodgins,J.K., Pollard, N.S.Interactive control of avatars animated with human motion data.*ACM Trans.Graph.21*, 3 (Jul.2002), 491–500.
8. Lee, Y., Lee, S.J., Popovic´, Z. Compact character controllers.In *SIGGRAPH Asia '09:ACM SIGGRAPH Asia 2009 Papers* (2009), ACM,New York, 1–8.
9. Levine, S., Wang, J.M., Haraux, A., Popovic´, Z., Koltun, V. Continuous character control with low-dimensional embeddings.*ACM Trans.Graph.31*, 4 (2012), 28.
10. Lo, W.Y., Zwicker, M. Real-time planning for parameterized human motion.In *SCA '08:Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008).
11. McCann, J., Pollard, N. Responsive characters from motion fragments.*ACM Tran.Graph.(SIGGRAPH 2007) 26*, 3 (Jul.2007).
12. Mount, D., Arya, S. Ann:A library for approximate nearest neighbor searching.1997. http://www.cs.umd.edu/~mount/ANN/.
13. Park, S.I., Shin, H.J., Shin, S.Y.On-line locomotion generation based on motion blending.In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2002), ACM, New York, 105–111.
14. Shin, H.J., Oh, H.S.Fat graphs:Constructing an interactive character with continuous controls. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2006), Eurographics Association, Aire-la-Ville, Switzerland, 291–298.
15. Sutton, R., Barto, A. *Reinforcement Learning:An Introduction*.MIT Press, Cambridge, MA, 1998.
16. Treuille, A., Lee, Y., Popovic´, Z. Near-optimal character animation with continuous control.*ACM Trans.Graph.26*, 3 (2007), 7.
17. Wang, J.M., Fleet, D.J., Hertzmann, A. Gaussian process dynamical models for human motion.*IEEE Trans.Pattern Anal. Mach.Intell.30*, 2 (2008), 283–298.
18. Ye, Y., Liu, K. Synthesis of responsive motion using a dynamic model. *Comput.Graph.Forum (Eurographics Proceedings) 29*, 2 (2010).
19. Zhao, L., Safonova, A. Achieving good connectivity in motion graphs. In *Proceedings of the 2008 ACM/ Eurographics Symposium on Computer Animation* (Jul.2008), 127–136.
20. Zordan, V.B., Majkowska, A., Chiu, B., Fast, M. Dynamic response for motion capture animation.*ACM Trans.Graph.24*, 3 (2005), 697–701.

Yongjoon Lee (yjlee@bungie.com), Bungie, Inc., Bellevue, WA.

Kevin Wampler 和 Jovan Popovic´ ({kwampler, jovan}@adobe.com), University of Washington, Adobe, Inc., Seattle.

Gilbert Bernstein 和 Zoran Popovic´ ({gilbo, zoran}@cs.washington.edu), University of Washington, Seattle.

译文责任编辑：周昆