第1章
八卦

马克·杰拉斯蒂

任何人都可以造谣，但没有人能阻止谣言。（美国谚语）

摘要流言在人类社会中起着非常重要的作用。 信息以惊人的速度在人类的小道消息中传播，通常在社区中几乎每个人都能得到信息，没有任何中央协调员。 此外，谣言往往非常顽固：一旦传播，几乎不可能消除它。 在许多分布式计算机系统中，特别是在云计算和对等计算中，这种速度和健壮性，加上算法的简单性和缺乏中央管理，都是非常有吸引力的特征。 因此，在过去的几十年里，已经开发了几种基于流言的算法来解决各种问题。 在这一章中，我们重点讨论了八卦的两个主要表现：信息传播（也称为组播），其中一条新闻正在传播；信息聚合(或分布式数据挖掘），其中分布式信息正在总结。 对于这两个主题，我们讨论理论问题，主要依赖于流行病学的结果，我们还考虑了分布式应用程序中的设计问题和优化。

目标

- 解释基于八卦的信息传播的基本属性
- 展示八卦方法如何用于另一个领域：信息聚合
- 讨论基于流言的示例系统，或者应用基于流言的组件

马克·杰拉斯蒂
Szeged大学和匈牙利科学院，H-6701Szeged，PO Box652.e-mail：
jelasity@inf.u-szeged.hu
 预印版：GiovannaDiMarzoSerugendo，Marie-PierreGleizes，和AnthonyKarageorgos，编辑，自组织软件：从自然到人工适应，自然计算系列，第139-162页，2011.DOI：10.1007/978-3-642-17348-6_7

1.1 导言

1.1.1 八卦

不管你喜不喜欢，流言蜚语在人类社会中扮演着关键的角色。 邓巴（一位人类学家）在他的有争议的书中，甚至声称语言出现的主要原因是允许流言蜚语，这不得不取代修饰-灵长类动物的一种常见的社会强化活动-由于早期人类群体规模的增加，在这些群体中，修饰不再是可行的[5]。

不管是什么情况，毫无疑问，流言蜚语——从主要的社会活动开始——在传播信息方面是非常有效的。 特别是，信息传播非常迅速，而且这一过程最能抵制阻止它的企图。 事实上，有时它会造成严重的损害，特别是对大公司。 有关某些公司与撒旦教有关联的谣言，或声称某些连锁餐厅出售含有鼠肉的汉堡或含有牛眼球液体的奶昔作为增稠剂等，并不罕见。 因此，控制流言一直是一个重要的研究领域。 金梅尔的这本书给出了许多关于人类八卦[15]的例子和细节。

虽然流言通常被认为是传播信息的一种手段，但在现实中，信息不仅是机械地传递的，而且也是经过处理的。 一个人收集信息，处理它，并传递处理后的信息。 在最简单的情况下，信息被过滤至少是为了它的兴趣程度。这将导致最有趣的新闻传播到整个群体，而不太有趣的新闻将在传播到每个人之前停止传播。 更复杂的场景也并不罕见，因为信息会逐渐改变。 这增加了过程的复杂性，并可能导致紧急行为，其中社区充当“集体智能”（有时可能不那么智能）信息处理介质。

1.1.2 流行病

流言类似于流行病，病毒扮演着一段信息的角色，感染扮演着学习信息的角色。 在过去的几年里，我们甚至不得不学习“病毒营销”等概念，这些概念是通过Web2.0平台（如视频共享网站）实现的，在这些平台上，广告商有意识地利用日益高效和扩展的社交网络通过流言传播广告。 关键的想法是，令人震惊或非常有趣的广告特别是de-

签署，以最大限度地增加观众通知他们的朋友的机会，等等。

毫不奇怪，流行病传播与流言蜚语有着相似的性质，同样（如果不是更多的话）对理解和控制也很重要。 由于这种类比和遵循共同的实践，我们将混合流行病学和流言术语，并将流行病传播理论应用于流言系统。

1.1.3 分布式系统的经验教训

流言蜚语和流行病对大规模分布式系统来说是有兴趣的，至少有两个原因。第一个原因是设计新协议的灵感：八卦有几个吸引人的特性，如简单性、速度、健壮性以及缺乏中央控制和瓶颈。 这些属性对于信息传播和集体信息处理（聚合）都是大规模分布式系统的关键组成部分非常重要。

第二个原因是安全研究。 随着互联网的稳步发展，病毒和蠕虫在传播策略上变得越来越复杂。 受感染的计算机通常组织成网络（称为僵尸网络），并且能够合作和执行协调攻击，它们对IT基础设施构成了非常重大的威胁。 打击这些网络的一种方法是设法防止它们传播，这就需要对互联网上的流行病有一个很好的了解。

在这一章中，我们将重点放在八卦和流行病的前一个方面：我们将它们视为设计健壮的自组织系统和服务的灵感。

1.1.4 大纲

我们讨论了两个应用领域的流言传播模型：信息传播（1.2节)和信息聚合(1.3节）。 对于这两个领域，我们首先介绍关键的方法和算法，以及它们的理论性质，在可能的情况下。 随后，我们在每个部分中讨论了这些想法的应用，在这些部分中，我们更关注实际的细节和设计问题。 在1.4节中，我们简要地提到了我们之前没有详细讨论的应用程序域，但这也是八卦的有希望的应用程序。 在第1.5和1.6节中，我们列出了一些关键的结论，并提出了几篇文章，读者可能会发现这些文章是有用的，值得仔细阅读。

1.2 信息传播

八卦（或流行病）在计算机系统中最自然的应用是传播信息。 在大规模分布式系统中，进程定期与同行通信和交换信息的基本思想并不罕见，从互联网的早期就得到了应用。 例如，Usenet新闻组服务器使用类似的方法传播帖子，IRC聊天协议在IRC服务器中也应用了类似的原则。 在许多路由协议中，我们还可以观察到路由器与相邻路由器通信和交换流量信息，从而改进路由表。

然而，第一个基于理论和仔细分析的流言的真正应用，促进了对流言协议家族的科学研究，是施乐公司分布式数据库系统的一部分，并被用来确保施乐内部网络上的数据库的每个副本都是最新的[4]。 在本节中，我们将使用这个应用程序作为一个激励示例和说明，同时介绍基于流言的信息传播算法的几个变体。

1.2.1 问题

让我们假设我们有一组数据库服务器（在施乐的例子中，有300个，但是这个数字也可能更大）。 所有这些服务器都接受更新；即新记录或修改现有记录。 我们希望将每次更新通知所有服务器，以便数据库的所有副本都是相同的和最新的。

显然，我们需要一个算法来通知所有服务器有关给定的更新。 我们将称之为任务更新扩展。 此外，我们还应该考虑到，无论我们使用什么算法来扩展更新，它都不会完美地工作，因此我们需要一个纠错机制。

在施乐，更新扩展最初是通过电子邮件发送更新到所有服务器来解决的，纠错是通过手工完成的。 发送电子邮件显然是不可伸缩的：发送节点是一个瓶颈。 此外，有多种错误来源是可能的：发送方可以在网络中有一个不完整的服务器列表，一些服务器可以暂时不可用，电子邮件队列可以溢出，等等。 这两个任务都可以使用适当的（单独的）流言算法以更可伸缩和可靠的方式解决。 在下面我们首先介绍了几种八卦模型和算法，然后我们解释了如何应用各种算法解决上述问题。

1.2.2 算法和理论概念

我们假设我们得到了一组能够相互传递消息的节点。 在本节中，我们将重点讨论在这些节点之间扩展单个更新。 也就是说，我们假设在某个时间点，其中一个节点从外部来源获得一个新的更新，从这一点出发，我们对该更新的传播动态感兴趣。

在讨论算法和理论模型时，我们将使用流行病学的术语。 根据这个术语，每个节点可以处于三种状态之一，即

- 敏感(S)：节点不知道更新
- 感染(I)：节点知道更新并正在积极传播它
- 删除(R)：节点已经看到更新，但不参与传播过程（在流行病学中，这相当于死亡或免疫）

这些状态相对于一个固定的更新。 如果有几个并发更新，一个节点可以被一个更新感染，同时仍然容易受到另一个更新的影响，以此类推。为了理论讨论的目的，我们将制定我们的算法，假设系统中只有一个更新，但假设节点不知道只有一个预期的更新。 这将使我们能够导出更新传播的关键理论特性，同时保持算法简单。

在实际应用程序中，通常会同时传播许多更新，并且不断插入新的更新。在这种情况下，可以应用额外的技术来优化传播单个更新的摊销成本。 在1.2.3节中，我们讨论了其中的一些技术。 此外，节点可能知道全局列表，甚至更新的插入时间，以及其他一些节点可用的更新列表。 这些信息也可以进一步降低传播成本。

允许的状态转换取决于我们研究的模型。 接下来，我们将考虑SI模型和SIR模型。 在SI模型中，节点最初处于状态S，可以更改为状态I。 一旦进入状态I，节点就不能再改变其状态（我是吸收状态）。 在SIR模型中，我们允许状态I中的节点切换到状态R，其中R是吸收状态。

1.2.2.1 SI模型

在SI模型中实现流言的算法如算法1所示。 它是以异步消息传递样式表示的，其中每个节点执行一个进程（我们称之为活动线程），此外，它还有处理传入消息的消息处理程序。

活动线程在每个Δ时间单元中执行一次。 我们将这个等待期称为流言周期（其他术语也被使用，如流言轮或周期）。

算法1SI八卦

```
1：循环
2：     等等（Δ）
3：     p←随机同行
4：     如果推和状态我
5：          发送更新到p
6：     结束如果
7：     如果拉的话
8：          发送更新请求到p
9：     结束如果
10：结束循环

11：程序ON UPDATE(M)
12：     存储m.更新        ◁ 意味着切换到状态
I13：结束过程
14：
15：程序ON UPDATER EQUEST(m)16：
          如果在我的状态下
17：          向发件人发送更新
18：     结束如果
19：结束程序
```

在第3行中，我们假设一个节点可以从所有节点的集合中选择一个随机的对等节点。 这个假设并不微不足道。 我们将在1.4节中简要讨论随机同行抽样。

该算法利用两个重要的布尔参数，称为推和拉。 其中至少有一个必须为真，否则不发送消息。 根据这些参数，我们可以谈论推、拉和推拉八卦，每一个都有显著不同的动态和成本。 在推送流言中，易感节点是被动的，感染节点主动感染人群。 在推拉八卦中各个节点都是活动的。

请注意，在第7行中，我们不测试节点是否被感染。 原因是我们假设节点不知道预期有多少更新，也不交换关于它们收到的更新的信息。 显然，一个节点不能停止拉更新，除非它知道可以期望什么更新；它也不能避免获得已知的更新，除非它广告它已经更新了。 因此，在最简单的情况下，我们一直在寻找任何更新。 优化以减轻这个问题是可能的，但以可伸缩的方式解决它并不是微不足道的。

为了理论上的目的，我们将假设消息是毫不延迟地传输的，目前我们将假设系统中没有发生故障。 我们还将假设消息在每个节点同时发送，即来自不同周期的消息不会混合，并且周期是同步的。 这些假设对于实际的可用性都不是至关重要的，但它们对于理论推导是必要的，尽管这些理论推导给出了八卦协议的定性和定量行为的一个公平的指示。

让我们从推送模式的讨论开始。 我们将考虑更新的传播速度作为节点数N的函数。 我们走吧$_{0}$ 表示在一个节点引入更新时易受影响节点的比例。 很明显$_{0}$ =(n-1)/n。 我们走吧$_{t}$ 表示t周期结束时易感节点的比例；即在时间tΔ. 我们可以计算s的期望$_{不+1}$ 作为a 职能 的 s_{t}, 已提供 那个 的 同伴 被选中了 线 3是 被选中了 独立的 在每个地方 节点 还有 独立的 的 过去 决定 作为好吧。 在 这个 案件， 我们 已经

$$E(s_{不+1}) = t\ 1 - \frac{1}{n}^{\sigma\ n(1-s_t)} \approx -^{(1-s_t)}, \quad (1.1)$$

其中N(1-s_t) 是 的 号码 的 节点 那个 都是 感染了 在 循环 t，还有 (1 -$1/n$) 是那个 概率 那个 a 固定的 感染了 节点 威尔 不是 感染 一些 固定的 易受感染 节点。 很明显， a 节点 是 易受感染 在 循环 t + 1 如果 它 是的 易受感染 在 循环 t 还有 全部 被感染的人 节点 被选中了 一些 其他 节点。 实际上， 作为 它 转身 出去， 这个 近似模型是相当准确的（偏离它很小），如 Pittel在[17]中所示：我们可以取预期值$E_{不+1}$作为一个很好的近似 $s_{不+1}$.

很容易看到，如果我们等待足够长的时间，那么最终所有节点都会收到更新。 换句话说，特定节点从未收到更新的概率为零。 但是，让每个节点知道更新（感染)所需的周期数呢）？ 皮特证明了可能性，

$$s_n = 日志_2\ N+日志N+O（1）作为N\rightarrow\infty, \quad (1.2)$$

在那里$_n$ =*分钟*{t：s_t =0}是传播更新所需的周期数。 证明是相当长和技术性的，但直观的解释是相当简单的。 在初始周期中，大多数节点都是易受影响的。 在这个阶段，感染节点的数量将在每个周期中增加一倍，达到一个很好的近似。 但是，在最后一个周期中，s_t 是小的，我们可以从（1.1）中看出 $E(S_{不+1}) \approx s_t e^{-1}$. 这个表明有一个第一阶段，持续大约对数$_2$ 循环，还有有一个最后的阶段持续日志N周期。 在这两个阶段之间的“中间”阶段可以被证明是非常快的，持续了恒定数量的循环。

方程（1.2）经常被引用为八卦被认为是有效的关键原因：它只需要O(logN)周期就更新通知每个节点，这表明了非常好的可伸缩性。 例如，在施乐的最初方法中，基于向每个节点发送电子邮件，所需时间为O(N)，假设电子邮件是按顺序发送的。

然而，让我们考虑在网络中发送的消息总数，直到每个节点被感染。 对于推送八卦，可以显示它是O(NlogN)。 直观地说，最后一个阶段持续O(logN)周期与s_t 做得很好 很小 已经 涉及 发送 也是 很多 信息 通过 的 感染了 节点。 大多数这些消息都是徒劳的，因为它们的目标节点已经被感染。 消息的最佳数量显然是O(N)，这是通过电子邮件方法获得的。

幸运的是，使用拉力技术可以显著提高推送方法的速度和消息复杂度。让我们考虑一下$_t$ 在拉八卦的情况下。 在这里，我们得到简单的公式 的

$$E(s_{不+1}) = i \bullet_t = s^2, \quad (1.$$

3) 如果我们假设方差，它直观地表示二次收敛
s_t 很小。 当s_t 很大，它会减少慢慢地。 在这一阶段，推动方法显然表现得更好。 但是， 当s_t 是小的，拉接近的结果是a

明显比推送更快的收敛速度。 其实二次收敛阶段，大致在s后，< 0.5，只持续O(log log N)周期，可以很容易地验证。 一个人当然可以把推拉结合起来。 这可以预期比单独推送或拉动更快地工作，因为在初始阶段，推送消息将保证快速传播，而在结束阶段，拉消息将保证在短时间内感染剩余节点。 虽然在实践中更快，但由于初始指数相位，推拉速度仍为O(logN。

消息复杂性呢？ 由于在每个周期中，每个节点将发送至少一个请求，而O(日志N)周期是更新到达所有节点所必需的，因此消息复杂度为O(N日志N)。 然而，如果我们只计算更新，而忽略请求消息，我们会得到不同的图片。 仅仅计算更新并不是没有意义的，因为更新消息通常比请求消息大数量级。 已经表明，事实上，推拉八卦协议只发送O(N日志N)更新的总[10]。

证明背后的基本思想再次是基于将扩展过程划分为阶段并计算每个阶段的消息复杂性和持续时间。 本质上，初始指数阶段-我们也看到了推送-只需要O(N)更新传输，因为感染节点（发送消息）的数量呈指数增长。 但最后一个阶段，二次收缩阶段，如拉所见，只持续O(logLogN)周期。 不用说，与其他理论结果一样，数学证明是相当涉及的。

1.2.2.2 SIR模型

在上一节中，我们给出了一些关于收敛速度和消息复杂性的重要理论结果。然而，我们忽略了一个在实际场景中可能很重要的问题：终止。

在SI模型中，推送协议永远不会终止，即使在每个节点收到每个更新之后，也会不断发送无用的更新。如果预先知道完整的更新列表，Pull协议可以停止发送消息：在接收到所有更新后，不需要发送更多的请求。 然而，在实践中，即使是拉协议也不能在SI模型中终止，因为更新列表很少为人所知。 在这里，我们将讨论SIR模型中终止问题的解决方案。这些解决方案总是基于某种形式的检测和作用的“年龄”的更新。

我们可以考虑两个不同的目标来设计我们的算法。 首先，我们可能希望确保终止是最优的；也就是说，我们希望通知所有节点更新，并且我们可能希望同时最小化冗余更新传输。 其次，我们可能希望选择一个不那么智能、简单的协议，并分析不会作为某些参数的函数获得更新的节点的比例的大小。

实现最优性的第一个设计目标的一个简单方法是显式地跟踪更新的年龄，并在预先指定的年龄时停止传输(即切换到移除的状态，从而实现SIR模型

算法2是SIR的八卦变体

```
1：循环
2：     等等（Δ）
3：     p←随机同行
4：     如果推和状态我
5：         发送更新到p
6：     结束如果
7：     如果拉的话
8：         发送更新请求到p
9：     结束如果
10：结束循环
11：
12：程序ON FEEDBA CK(M)
13：     用Prob切换到状态R。 1/k
14：结束程序
15：程序ON UPDATE(M)16：
        如果处于I或R状态
17：        向发件人发送反馈
18：    其他
19：        存储m.更新                ◃ 现在状态
I20：   结束如果
21：结束程序
22：
23：程序ON UPDATER EQUEST(M)24：
        如果在我的状态下
25：        向发件人发送更新
26：    结束如果
27：结束程序
```

已到达。 这个年龄阈值必须计算为最优的给定网络大小N使用上述理论结果。 当然，这假设每个节点都知道N。 此外，还假定了一个实际的无误差和无延迟传输，或者至少需要一个良好的实际传输误差模型。

除了这个问题，跟踪更新的时代明确地代表了另一个非平凡的实际问题。 在我们的理论讨论中，我们假设消息没有延迟，并且周期是同步的。当这些假设被违反时，很难以可接受的精度确定更新的年龄。

从这一点开始，我们将放弃这种方法，并将重点放在简单的异步方法上，这些方法更健壮和一般，但不是最优的。 为了实现简单性和合理性能相结合的第二个设计目标，我们可以尝试根据本地信息和可能从少数同行收集的信息来猜测何时停止。 这些算法具有简单性和局部性的优点。 此外，在SIR模型的许多应用中，没有必要像我们稍后将看到的那样对完全传播进行强有力的保证。

也许最简单的实现是当节点在遇到已经收到更新的对等点时以固定的概率移动到被移除的状态。 让这个概率为1/k，其中参数k的自然解释是一个节点将更新发送给一个在停止传输之前已经有更新的对等点的平均次数。显然，这隐含地假设了一个反馈机制，因为节点需要检查他们发送更新的对等点是否已经知道更新。

如算法2所示，这种反馈机制是SIR和SI八卦的唯一区别。 活动线程和过程ONUPDATEREQUEST与算法1相同。 然而，当已知接收到的更新时，过程ON UPDATE发送反馈消息。 此消息由过程ONFEEDBA CK处理，最终将节点切换到已删除的状态。 当处于移除状态时，程序ONUPDATEREQUEST将不再交付更新。

SIR算法的数学模型比SI模型复杂。 一个典型的方法是处理微分方程，而不是我们以前应用的离散随机方法。 让我们通过对算法2的分析来说明这种方法，假设一个推送变体。 下面[2,4]，我们可以写

$$\frac{DS}{DT} = -\text{内} \tag{1.4}$$

$$\frac{di}{DT} = \text{是} - \frac{1}{k}(1-s \tag{1.5}$$

其中s(T)和i(T)分别是易感节点和感染节点的比例。 被移除状态下的节点由r(t)=1-s(t)给出，i(t)我们可以取比，消除t：.

$$\frac{di}{DS} = -\frac{k+1}{k} + \frac{1}{KS}, \tag{1.6}$$

它的产量

$$\text{一} = -\frac{k+1}{k}S + \frac{1}{k}\text{日志}s+c, \tag{1.7}$$

其中c是积分常数，可以使用i(1-1/N)=1/N(其中N是节点数)的初始条件来确定)。 对于大的N，我们有c≈(k+1)/k。

现在我们对值s感兴趣* 我在哪里*=0：当时发送更新被终止，因为所有节点都是敏感的或删除的。 换句话说， s^* 是 的 比例 的 节点 那个 是的 不是 知道 的 更新 什么时候 八卦 停下来。 理想情况下，S* 应该是零。 利用结果，我们可以为s写一个隐式方程* 作为 以下：

$$= \exp[-(k+1)(1-s)]。 \tag{1.8}$$

这告诉我们传播是非常有效的。 对于k=1，20%的节点预测会错过更新，但对于k=5，0.24%会错过，而对于k=10，则只有0.00017%。

现在让我们开始讨论消息的复杂性。 由于总体上没有实现充分的传播，我们的目标现在是近似的消息数量，以减少敏感节点的比例到指定的水平。

让我们首先考虑推式变体。 在这种情况下，我们做了一个相当引人注目的观察，即s的值只取决于节点发送的消息m的数量。 事实上，每个受感染的节点随机地独立地选择对等节点来发送更新。 也就是说，每个更新消息都被发送到一个独立于所有节点集随机选择的节点。 这意味着固定节点在发送了总共m个更新消息后处于状态S的概率可以近似为

$$= (1-\frac{1}{n})^m \approx \exp[-\frac{m}{n}] \tag{1.9}$$

代入期望值s，我们可以很容易地计算出系统中需要发送的消息总数：它是

≈-N个日志 (1.10)

如果我们要求s=1/N，即我们只允许单个节点不查看更新，那么我们需要m≈N日志N。 这让我们想起了SI模型，它具有O(NlogN)消息复杂性，以实现完全传播。 另一方面，如果我们允许一个恒定比例的节点没有看到更新(s=1/c)，那么我们有m≈N个日志c；也就是说，一个线性数目的消息就足够了。 注意，不能直接设置s或m，只能通过k等其他参数设置。

另一个值得注意的一点是，（1.9）不管我们是否应用反馈机制，也不管应用于切换到状态R的精确算法。 事实上，它甚至适用于纯SI模型，因为我们只假设它是一个带有随机同行选择的推送式八卦。 因此，它是一种非常简单的替代方法来说明SI模型所显示的O(NlogN)消息复杂性结果：粗略地说，我们需要大约N个logN消息才能使s低于1/N。

由于m决定s，而不考虑应用的推送八卦算法的细节，因此算法能够使感染节点发送m消息的速度决定了s的收敛速度。 考虑到这一观察，让我们比较SIR流言的一些变体。

除了算法2之外，还可以通过几种不同的方式实现终止(切换到状态S。 例如，与其在过程ONFEEDBA CK中进行概率决策，还可以使用计数器，并在接收到第k反馈消息后切换到状态S。 反馈可以完全消除，移动到状态R只能取决于节点发送更新的次数。

不难看出计数器变体改善了负载平衡。 这反过来提高了速度，因为如果消息发送负载很好地平衡，我们总是可以在固定的时间内发送更多的消息。事实上，在上述变体中，应用没有反馈的计数器会导致最快的收敛。 然而，必须适当地设置参数k以达到所需的s级才能适当地设置k和s，您需要知道网络大小。 . 使用反馈机制的变体实现了稍微不那么有效的负载平衡，但它们对k值和网络大小更健壮：它们可以根据反馈“自我调整”消息的数量。 例如，如果网络很大，在收到第一次反馈之前，更多的更新消息将是成功的。

最后，与SI模型一样，很明显，在结束阶段，拉变量要快得多，并且使用更少的更新消息。 它这样做的代价是不断发送更新请求。

我们通常认为，特别是当更新不断被注入时，具有计数器和反馈的推拉算法可能是最理想的选择。

1.2.3 申请

我们首先解释了我们讨论的各种协议是如何在施乐应用于维护数据库的一致副本集的。 虽然我们不能在这里提供完整的图片（见[4]），但我们阐明了最重要的想法。

在1.2.1节中，我们确定了两个子问题，即更新扩展和纠错。 前者由SIR流言协议实现，后者由SI协议实现。 SIR的流言蜚语被称为谣言传播，当一个新的更新进入系统时就会运行。 请注意，在实践中，许多新的更新可以捎带一个八卦消息，但上述收敛属性适用于任何单个固定更新。

用于纠错的SI算法适用于每次输入的更新，而不考虑年龄，同时适用于所有更新。 在天真的实现中，整个数据库将在每个周期中由每个节点传输。显然，这不是一个好主意，因为数据库可能非常大，而且大多相当相似。相反，节点首先试图通过交换压缩的描述（如校验和(或在不同时间取的校验和列表）来发现它们的本地副本之间的区别，并且只发送丢失的更新。然而，一个周期的纠错通常比造谣要贵得多。

用于纠错的SI算法称为反熵。 这不是一个非常幸运的名字：我们应该在这里指出，它没有更深的含义，而是表达这样一个事实，即“反熵”将增加副本之间的相似性，从而增加“顺序”（减少随机性）。 因此，由于熵通常被认为是“无序”的一种度量，所以“反熵”这个名字在这种情况下只是指“反无序。

在完整的系统中，新的更新通过谣言传播，反熵偶尔运行，以处理任何未交付的更新。 当找到这种未交付的更新时，通过将给定的更新作为新的更新重新插入到不存在的数据库中来重新分配给定的更新。 这是一个非常简单和有效的方法，因为通过谣言传播的更新传播的成本取决于已经有更新的其他节点的数量：如果大多数节点已经有了它，那么重新分配将很快消失。

让我们快速地将这个解决方案与早期的基于电子邮件的方法进行比较。邮件更新和谣言传播是相似的，因为两者都专注于传播单个更新，并且有一定的小错误概率。 与电子邮件不同，流言没有瓶颈节点，因此对本地故障不太敏感，对带宽等本地资源的假设较少。 这使得流言蜚语成为一种更可扩展的解决方案。 流言总使用稍微多一点的消息来分发一个更新。 但是随着大量副本的频繁更新，流言的摊余成本（每次更新的消息数量)更有利(请记住，一条消息可能包含许多更新）。

在实际实现中，已经执行了额外的重要优化。 也许最有趣的是空间八卦，在空间八卦中，节点不是随机选择对等点，而是根据距离度量来选择对等点。 这一点很重要，因为如果底层物理网络拓扑结构如此之大，就会出现瓶颈

连接密集集群的链路，然后随机通信将大量负载放置在这些链路上，这些链路随系统大小呈线性增长。 在空间八卦中，节点倾向于拓扑中更接近的节点，从而减轻了长距离链路的负载，但同时牺牲了一些扩展速度。 本专题在[12]中作了很长的讨论。

我们还应该提到删除数据库条目。 这是通过“死亡证书”来解决的，这些证书是说明应该删除给定条目的更新。 不用说，死亡证书不能无限期地存储，因为最终数据库会被他们超载。 这个问题需要额外的技巧，例如删除大部分但不是全部，这样如果删除的更新再次弹出，就可以重新激活死亡证书。

除了上面讨论的应用程序之外，流言范式最近又得到了另一个提升。 在习惯了Grid和P2P应用程序，并见证了巨大的、通常是地理分布的数据中心的出现，这些数据中心的规模和容量以令人难以置信的速度增长之后，在过去的几年里，我们不得不学习另一个术语：云计算[7]。

云计算涉及大量的分布式资源（云），通常由一个组织拥有，并以这样的方式组织，对用户来说，它似乎是一个连贯和可靠的服务。 例如存储服务或网格式计算服务。 最近，IT行业的大公司推出了自己的云计算解决方案，例如谷歌和IBM[16]以及亚马逊[1]。

这些应用程序在编写时代表了尖端技术，由于公司保密，它们如何工作并不总是很清楚，但从几个来源来看，似乎相当明显地涉及流言协议。 例如，在亚马逊的S3存储服务最近崩溃之后，解释失败的消息包括一些细节：

> （.）Amazon S3使用八卦协议在整个系统中快速传播服务器状态信息。 这允许Amazon S3在失败或无法到达的服务器上快速路由，等等。[1] (...)

此外，最近一份关于亚马逊计算体系结构基础技术的学术出版物提供了关于流言协议[3]的进一步细节，揭示了反熵流言协议负责在每台服务器上维护一个完整的成员表（即一个具有服务器状态信息的完全连接的覆盖网络）。

1.3 汇总

八卦传播范式可以推广到信息传播以外的应用领域。 在这些应用中，一些隐含的传播概念

[1] http://status.aws.amazon.com/s3-20080720.html

信息仍然存在，但重点不仅在于传播，而且还在于处理信息。

这种处理可以用于创建分布式数据的摘要；也就是说，仅基于八卦式通信，在节点集上计算全局函数。 例如，我们可能对节点的某些属性的平均值或最大值感兴趣。 计算这种全局函数的问题称为数据聚合或简单聚合，我们也可能希望计算更复杂的函数，例如在完全分布式数据上拟合模型，在这种情况下，我们讨论分布式数据挖掘问题。.

在过去的几年里，许多努力都是针对一个具体的问题：计算平均数。 平均化可以被认为是聚合的典型例子。 这是一个非常简单的问题，但非常有用：基于适当定义的局部属性的平均值，我们可以计算出广泛的值。 为了精心-

关于这个概念，让我们介绍一些形式主义。 让你$_{i}$ 做一个 σ 属性值在节点I代表所有0<我≤N。 我们对平均y¯=感兴趣 $\sum_{i=1}^{n} y_i/n$。 如果我们可以计算平均值，然后我们可以计算形式的任何平均值

$$g(z_1, z_n)=f^{-1}\left(\frac{\sum_{i=1}^{n} f(z_i)}{n}\right), \tag{1.11}$$

其中f()是一个合适的函数。 例如，f(X)=logx生成几何平均值，而f(X)=1/x生成谐波平均值。 很明显，如果是的话$_{i}$ = f (z$_{i}$) 然后我们可以很容易地计算g(Z$_{1}$，. . . ，z_n)=f $^{-1}$(y¯)。 此外，如果我们计算y的几个幂的平均值$_{i}$然后，我们可以计算出值分布的经验矩。 例如，方差的（偏）估计可以表示 作为a 职能 结束了 平均数 的 y^2 还有 y_i:

$$\hat{c}_n^2 = \bar{y^2} - \text{你}^2 \tag{1.12}$$

最后，可以使用平均作为原语来计算其他有趣的量。 例如，如果每个属性值为零，除了在一个节点上，其中值为1，则y¯=1/N，因此网络大小由N=1/y¯给出。

在本节的其余部分，我们重点讨论了几种用于计算节点属性平均值的八卦协议。

1.3.1 算法和理论概念

我们讨论的第一个，也许是最简单的算法是推挽平均，在算法3中提出。

每个节点周期性地选择一个随机对等点进行通信，然后发送平均x的局部估计。 然后，接收方节点用自己的当前估计来回复。 两个参与节点（发送方和发送应答的节点）都将存储前两个估计值的平均值作为新的估计值。

算法3推拉平均

1：循环
2: 等等（Δ）
3: p←随机同行
4: 发送推送(X)到p
5：结束循环

6：程序ON PU SHU PDATE(M)7：
将拉力(X)发送到发送器
8: *x←(m. x+x)/2*
9：结束程序
10:
11：程序启动(M)12：
x←(m. x+x)/2
13：结束程序

与我们对信息传播的处理类似，算法3是为异步消息传递模型制定的，但在讨论算法的理论行为时，我们将假设几个共时性质。 我们将回到1.3.1.1节中的异步问题。

目前，我们还将该算法视为一次算法；也就是说，我们假设首先是局部估计x_i 节点i初始化为x_i =$_i$ 对于所有节点，我=1。 . . N，然后执行八卦算法。 这一假设也将在1.3.1.2节中放宽，在这里我们简要讨论了属性y的情况$_i$ 可以随着时间的推移而改变，任务是不断更新平均值的近似值。

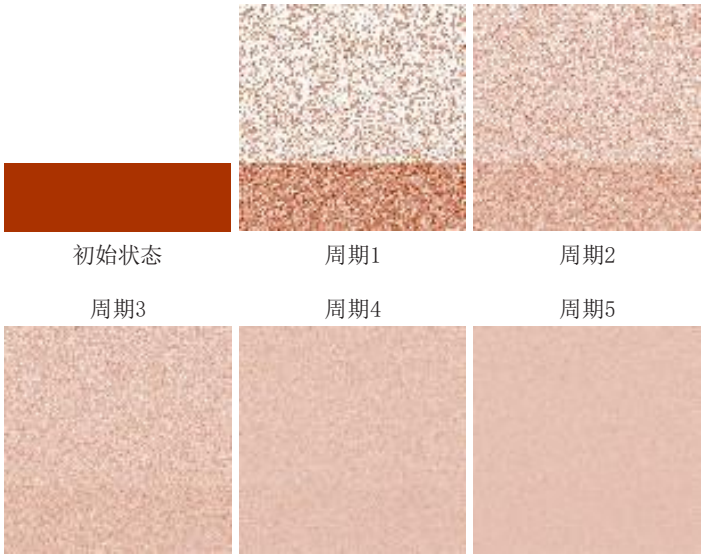
让我们先看看算法的收敛性。 很明显，当x时的状态$_i$ =你¯ 假设没有节点故障和消息故障， 还有 那个 的 信息 都是 已交付 没有 延迟。 它 是 不是 非常 很难 去说服 自己 那个 算法 收敛 为了这个 固定的 点 在概率上。 我们 省略 的 技术方面 细节 的 的 证明； 的 恶作剧 是 去 首先 观察 那个 的 *和* 的 整个近似保持不变。 更多 准确地说，

$$\sum_{n} x_i(t) = \sum_{i=1}^{n} x_i(0) \tag{1.13}$$

任何t。 *这个非常重要的性质叫做质量守恒。*然后，我们研究了最小近似和最大近似之间的差异，并表明这种差异只能减少，而且，它收敛到零的概率，使用的事实是同行是随机选择的。 但如果所有的近似都是相同的，由于质量守恒，它们只能等于y¯。

然而，真正有趣的问题是收敛的速度。 收敛性的事实在概率意义上很容易证明，但从实际的角度来看，这样的证明是无用的，而不能描述速度。协议的速度如图1.1所示。 这一过程表现出类似扩散的行为。 回想一下，扩散是在像素上的随机网络上执行的，而不是在二维网格上执行的。 像素的排列仅用于插图目的。

通过研究类方差统计量可以表征收敛速度



无花果。 1.1说明平均协议。 像素对应于节点(100x100像素=10，000个节点)，像素颜色对应于平均值的局部近似。

$$\zeta = \frac{1}{n}\sum_{i=1}^{n}(x_i - y)^2, \tag{1.14}$$

它描述了当前估计集的准确性。 可以显示(见[11,9]

$$E(\sigma_n^2(t+1)) \leq \frac{1}{2}\sigma_n^2(t), \tag{1.15}$$

其中时间索引t表示周期。 也就是说，方差在每个周期中减少一个常数因子。 在实践中，协议的10-20个周期已经提供了一个非常精确的估计：协议不仅收敛，而且收敛速度也非常快。

1.3.1.1 不同步

在信息传播的情况下，允许不可预测和无界消息延迟（异步模型的一个关键组件)对协议的正确性没有影响，它只对传播速度有(在实践中是边际的）影响。 然而，对于算法3，在存在消息延迟的情况下，正确性不再得到保证。

要了解原因，假设节点j从节点i接收到PUSHUPDATE消息，因此它修改了自己的估计值，并将自己以前的估计值发回i。 但在这一点之后，网络的质量守恒性质将

算法4推平均

1：循环
2: 等等（Δ）
3: p←随机同行
4: 发送(x/2，w/2)到p
5: $x \leftarrow x/2$
6: $w \leftarrow w/2$
7：结束循环

8：程序ON UPDATE(M)9：
$x \leftarrow m.x + x$
10: $\leftarrow m.w + w$
11：结束程序

被违反：所有近似的总和将不再正确。 如果节点j和节点i都没有在等待回复的时间节点期间接收或发送另一条消息，则这不是问题。 然而，如果他们这样做了，那么网络的状态可能会被破坏。 换句话说，如果对推送和拉消息不是原子的，异步是不能容忍的。

算法4是算法3的一个巧妙的修改，对消息延迟具有更强的鲁棒性。 算法非常相似，但在这里我们引入了另一个属性w。 对于每个节点i，我们最初设置$w_i = 1$（所以的和的这些价值是n）。 我们也是修改的口译的的电流估计数：上节点i它威尔是x_i/w_i而不是x_i就像推拉一样变体。

为了理解为什么这种算法对消息延迟更健壮，请考虑我们现在在不同的意义上具有质量守恒：对于属性x和w，节点处属性值的和加上未传递消息中属性值的和保持不变。 这很容易看出是否考虑在本地保存一半值并在消息中发送另一半的活动线程。 此外，还可以证明近似x的方差$_i/w_i$只能减少。

因此，消息现在可以延迟，但是如果消息延迟是有界的，那么节点和等待传递的消息的近似集的方差将趋于零。 由于质量守恒，这些近似将收敛到真正的平均值，而不管未传递消息中的总“质量”有多少。（注意x的方差$_i$或者w_i一个人不能保证收敛零。）

1.3.1.2 对失败和活力的强大

我们现在将考虑消息和节点故障。 不幸的是，这两种故障比异步故障更有问题。 在信息传播的情况下，失败对正确性没有影响：消息失败只会减缓传播过程，只有当存储新更新的每个节点失败时，节点失败才有问题。

在推送平均的情况下，丢失消息通常会破坏质量守恒。 在推拉平均的情况下，丢失推送消息将没有影响，但丢失回复（拉消息）可能会破坏质量守恒。 这个问题的解决方案要么基于故障检测(也就是说，它们假设一个节点

能够检测消息是否已传递)并根据检测到的故障纠正操作，或者它们基于一种形式的恢复（重新启动），其中协议定期重新初始化估计，从而恢复总质量。 由于协议的快速收敛，重新启动解决方案是可行的。 这两种解决方案都有点不优雅；但流言之所以吸引人，主要是因为缺乏对故障检测的依赖，这使得重新启动更符合整体流言设计理念。 不幸的是，由于消息故障，重新启动仍然允许有界的不准确，而故障检测提供了精确的质量守恒。

节点故障也是问题的根源。 节点故障是指节点离开网络而不通知其他节点的情况。 因为电流近似x_i（或 x_i/w_i）的 a 失败了 节点 i 是 通常 与 y不同，剩下的一套 节点 最终会有一个不正确的结果 近似 剩余属性值的平均值。 处理节点故障甚至是有问题的 如果 我们 假设 完美 失败 探测器。 解决办法 通常 参与 节点 分别存储每个节点的贡献。 例如，在推拉平均协议中，节点I 会储存 δ_{ij}节点j对x的增量贡献之和。 更准确地说，当从 j（推或拉）接收更新时，节点i
计算 $\delta_{ij} = \delta_{ij} + (x_j - x_i)/2$. 当节点i检测到节点j失败时，它执行校正 $x_i = x_i - \delta_{ij}$.

我们应该提到，只有当选定的节点来自一个小的固定的相邻节点集（而不是从网络中随机选择）时，这是可行的，否则所有节点都需要监视过多的其他节点以防止故障。 此外，消息失败也会干扰这个过程。 由于节点暂时失败，情况更加复杂，甚至可能没有意识到一些节点已经很长时间无法访问它们。 还请注意，重新启动方法也解决了节点故障问题，没有任何额外的努力或故障检测器，尽管如前所述，允许一些不准确。

最后，让我们考虑一个动态场景，其中由于y值的变化而违反了质量守恒(因此在节点上演化的近似将不再
反映正确的平均值)。 在这种情况下，我们可以简单地设置$x_i = x_i + y_i^{new} - y_i^{old}$，
这纠正了近似的总和，尽管协议需要一些
是时候再次收敛了。 与以前的情况一样，在没有任何额外措施的情况下，重新启动也解决了这个问题。

1.3.2 申请

我们所关注的基于扩散的平均协议最常被用作原语，以帮助其他协议和应用，如负载平衡、任务分配或计算相对复杂的分布式数据模型，如底层图形[8,13]的光谱特性。 传感器网络是应用程序特别有趣的目标，因为它们的目的是数据聚合，而且它们本质上是本地的：节点通常可以与之通信

他们的邻居只[19]。 然而，传感器网络不支持任意对节点之间的点对点通信，这使得平均速度要慢得多。

作为聚合的一个具体应用，我们更详细地讨论了一个名为Astrolabe[18]的中间件系统，据传该系统在亚马逊以某种形式使用。 星盘都是关于聚集的，但基本思想不是扩散。 相反，聚合是沿着虚拟层次结构传播的。 然而，正如我们将看到的那样，Astrolabe在许多方面都是基于流言蜚语的，它是我们所意识到的聚合思想的最“真实世界”应用，因此在它上投入几段是有用的。

让我们从Astrolabe基于的层次结构开始。 首先，每个节点被分配一个类似于域名的名称（连接时），域名由区域组成。 例如，一个节点可以称为“/Hungary/Szeged/pc3”，它假定三个区域：根区“/”和三个连续的更具体的区域。

每个区域都有一个描述符：属性列表。 叶区(例如，“pc3”)在其描述符中具有原始系统属性，如存储空间、存储文件等。 这些属性将是聚合的输入。 较高级别区域的描述符包含较低级别区域的摘要；因此，根区域描述符包含系统宽聚集体。

关键的思想是，区域描述符的新副本保持在属于给定区域的节点上，以及在作为给定区域的兄弟姐妹的节点上。例如，保持前面的示例，区域“Szeged”的描述符将在同一区域的所有节点上复制，此外，在“匈牙利”下的区域中的所有节点上复制”。 这样，“匈牙利”下的任何节点都可以计算“匈牙利”的描述符（它们都将拥有“匈牙利”下所有区域的描述符的副本”）。 “匈牙利”的描述符将被复制到根区下的每个区域，等等。 最终每个节点都能够计算根区域的描述符。

现在应该清楚的是，实现可以归结为复制数据库，正如我们在前面关于信息传播的章节中所看到的那样，因为每个节点都有一组需要复制的区域描述符。 事实上，Astrolabe为此目的使用了反熵的版本；只有情况更微妙，因为并不是每个节点都对所有描述符感兴趣。 正如我们所解释的，每个节点只对其自身区域的描述符(例如，“/”、“匈牙利/”、“Szeged/”和“pc3/”)以及作为其自身区域的兄弟姐妹的描述符感兴趣。 这样，系统可以避免复制潜在的巨大的完整描述符集，同时仍然能够在本地计算自己区域的描述符。

这种扭曲有两个重要的含义：第一，并不是所有节点都需要所有其他节点的列表，第二，节点可以从层次结构中更接近它们的节点中学习更多信息，这样它们就会希望更频繁地与这些节点通信。 这需要一种非平凡的对等选择方法。 Astrolabe本身使用描述符本身来存储所有感兴趣区域的联系人。然后，这些联系人可以用来闲谈（运行反熵超过两个节点的共同利益）。这样做的好处是，只要一个节点至少有一个来自任何一个节点的联系人

算法5推挽八卦算法骨架。

```
1：循环
2:     等等（Δ）
3:     ←选择对等()
4:     将推送（状态）发送到p
5：结束循环
6：程序启动(M)
7:     将拉力（状态）发送到发件人
8:     状态←更新(状态，m.state)
9：结束程序
10:
11：程序启动(M)12：  状态←更新(状态，m.state)
13：结束程序
```

它感兴趣的区域（也可以是根区域），它最终将自动填写每个相关区域的联系人列表。

与往常一样，有许多实际问题，如检测失败的节点（和区域）、连接过程以及其他涉及异步的细节、更新的时间戳等等。 这些可以在不牺牲系统的主要特性的情况下解决，尽管代价是失去了它最初的一些简单性。

最后，我们应该指出，Astrolabe是一个非常通用的系统，它支持的不仅仅是平均计算。 实际上，它支持一种类似SQL的语言，用于虚拟数据库上的查询，该查询由系统中所有区域的描述符集定义。 Astrolabe的行为就好像它是一个完整的数据库，而实际上，节点只复制一组完整数据中的一小部分（大致对数），并通过渐进摘要模拟一个完整的数据库。

1.4 八卦到底是什么？

到目前为止，我们已经讨论了八卦思想的两个应用：信息传播和聚合。 到目前为止，应该相当明显的是，这些应用程序虽然在细节上有所不同，但具有共同的算法结构。 在这两种情况下，活动线程都选择要通信的对等节点，然后是消息交换和两个节点（用于推拉)或一个节点(用于推拉)的内部状态的更新）。

很难理解这个“流言蜚语”概念的确切含义。 已经作出了这样做的努力，取得了喜忧参半的[14]。 例如，与随机同行的周期性和局部通信似乎是一个核心特征。 然而，在SIR模型中，节点可以停止通信。 此外，我们还看到，在一些流言协议中，相邻节点在每个周期中不需要是随机的，而是可以是固定的和静态的。 试图捕捉任何其他方面似乎会导致类似的哲学问题：任何涵盖我们直观地认为八卦的所有协议的特性似乎都涵盖了几乎所有分布式协议。 另一方面，限制总是排除协议，我们绝对认为流言蜚语。

因此，我们不打算描述这种常见的结构，而是提出算法5中显示的模板(或设计模式。 这个

模板涵盖了我们前面介绍的两个示例。 在信息传播的情况下，节点的状态是由存储的更新定义的，而在平均状态的情况下，状态是节点上平均值的当前近似。 此外，该模板还涵盖了大量其他协议。

本章没有讨论的一个值得注意的应用是覆盖网络的构建和管理。 在这种情况下，节点的状态是定义覆盖网络的一组节点地址。 （节点能够依赖网络堆栈的低层向地址发送消息；因此名称为“覆盖”。） 简而言之，状态（覆盖链接的集合）然后通过八卦进行通信，节点从他们所看到的所有链接的集合中选择他们的新链接集。 通过这种机制，可以创建和管理许多不同的覆盖网络，如随机网络、结构化网络（如环）或基于某种距离度量的邻近网络，例如基于语义或延迟的网络距离。

这些网络可以通过更高级别的应用程序或其他流言协议来应用。 例如，随机网络对于实现随机对等点抽样是非常好的，在本章中，所有算法在选择与之通信的随机对等点时都依赖于这种服务。

除了覆盖管理外，其他应用还包括负载均衡、资源发现和分配等。

1.5 结论

本章讨论了八卦传播模型的两个应用：信息传播和聚合。 我们表明，这两个应用程序都使用非常相似的通信模型，这两个应用程序都为高效和有效的执行提供了概率保证。

我们应该强调，流言协议代表了对分布式计算中“经典”方法的背离，在这种方法中，正确性和可靠性是重中之重，性能（特别是速度和响应能力）是次要的。 简单地说：流言协议-如果做得很好-是简单的，快速的，廉价的，和非常可扩展的，但它们并不总是在所有情况下和所有系统模型中提供完美或正确的结果。 但在许多情况下，“足够好的”结果是可以接受的，在现实的系统中，流言组件总是可以得到更重、但更可靠的方法的支持，从而提供最终的一致性或正确性。

一个相关的问题是恶意行为。 不幸的是，具有多个管理域的开放系统中的流言协议相当容易受到恶意行为的影响。 目前流言的应用集中在单个管理域系统上，其中所有节点都由同一个实体控制，因此问题的唯一来源是硬件、软件或网络故障。 在开放的系统中，节点可能是自私的，甚至更糟糕的是，它们可能是恶意的。 目前安全的流言-

算法比基本版本更复杂的数量级，从而失去了闲聊的许多原始优势。

总之，八卦算法是在一定假设下解决某些类型非常重要问题的伟大工具。特别是，它们可以帮助构建巨大的云计算系统，这些系统被许多人认为是未来的计算平台，并为传感器网络的编程提供工具。

关键点

- 流言协议是基于周期性的信息交换，在此期间，所有节点都与随机选择的对等节点通信。 这种八卦通信模型支持实现信息传播、聚合或覆盖拓扑结构等多个功能。
- 八卦协议代表了一种概率心态，我们在这里交换de-成功的术语保证（用强大的概率保证代替它）大大增加灵活性和可伸缩性。
- 基于推挽八卦的广播更新可以达到时间COM-O(logN)的丛集性和O(NlogLogN)的消息复杂度，接近基于树的广播的性能，同时具有极强的鲁棒性和对良性故障的可扩展性。
- 流言协议是新兴云计算的关键组成部分系统，以及传感器网络和其他完全分布式动态系统，这使得它们在实际应用中具有高度相关性。

致谢在撰写本章时，Jelasity得到了匈牙利科学院博莱伊奖学金的支持。

问题-练习

1.1. 我们已经看到，在一些协议中，跟踪更新的时代是有用的；也就是说，知道在多长时间前插入了给定的更新。 如果时钟是同步的，这很容易：一个人只需要一个时间戳。 试着考虑我们讨论的各种算法的算法，以保持跟踪或至少接近更新的年龄，如果时钟不同步！ 为了使问题更加困难，假设循环也不是同步的（也就是说，所有节点都有相同的周期Δ，但它们在随机时间开始循环）。 如果消息具有随机不可预测和无界延迟，您能做些什么吗？

1.2. 我们假设更新意外到达，没有更新的节点不知道它们错过了它。如果我们知道预期的更新列表，会有什么不同？ 推、拉和推拉有哪些优化是可能的？ 我们可以达到什么样的运行时间和消息复杂性？

1.3. 我们讨论了在强同步假设下传播的速度：我们假设周期是同步的，消息是在没有延迟或错误的情况下传递的。 考虑一下如果我们放弃同步假设的传播速度。 是慢还是快？

1.4. 我们说，如果消息可以延迟，那么推拉平均就会损坏。 想出一个例子来证明这一点！

1.6 进一步阅读

基于八卦的网络。 很好地描述了该领域在出版之日的艺术状况。 主要包含概述和立场文件演变在洛伦兹中心研讨会在莱顿，一个丰富的参考和想法来源。 (A.-M.Kermarrec和M.van Steen，编辑，ACM SIGOPS操作系统评论41，2007年。)

疫算法进行复制数据库维护。 这是一个引文经典，被认为是启动这一研究领域的工作。 它仍然非常值得一读；我们的讨论部分是基于这篇论文。(Demers，A.，Greene，D.，Hauser，C.，Irish，W.，Larson，J.，Shenker，S.，Sturgis，H.，Swinehart，D.，and Terry，D.，In：Proceedings of the6Annual ACM研讨会on Principles of分布式计算(PODC ‘87)，pp。 1－12. ACM出版社，1987年。)

传染病的数学理论及其应用。 一本旧书，对于相关的理论方面仍然是一个很好的起点。 (n。 t。 j。 贝利，格里芬，伦敦，第二版，1975年。)

参考资料

1. 亚马逊网络服务：http://aws.amazon.com
2. 贝利，N.T.J。 *传染病数学理论及其应用，第二版。 格里芬，伦敦（1975年）
3. De Candia，G.，Hastorun，D.，Jampani，M.，Kakulapati，G.，Lakshman，A.，Pilchin，A.，Sivasubramanian，S.，Vosshall，P.，Vogels，W.：Dynamo：Amazon的高可用的键值商店。 In：SOSP ‘07：第二十一届ACM SIGOPS操作系统原则研讨会论文集，pp。 205－220. ACM，纽约，纽约，美国（2007年）。 DOI10.1145/1294261.1294281
4. Demers，A.，Greene，D.，Hauser，C.，Irish，W.，Larson，J.，Shenker，S.，Sturgis，H.，Swinehart，D.，Terry，D.：用于复制数据库维护的流行病算法。 In：第六届ACM分布式计算原理研讨会论文集(PODC ‘87)，pp。 1－12. 加拿大不列颠哥伦比亚省温哥华ACM出版社（1987年）。 DOI10.1145/41840.41841

5. 邓巴，R：混乱，流言和语言的进化。 哈佛大学出版社（1998年）
6. Golub，G.H.，Van Loan，C.F.：矩阵计算，第三版。 约翰霍普金斯大学出版社（1996年）
7. 手，E：头在云端。 自然449，963（2007年）。 DOI10.1038/449963a
8. Jelasity，M.，Canright，G.，Engø-Monsen，K.：异步分布式功率迭代，基于八卦的规范化。 In：A.M.Kermarrec，L.Bouge，T。*Priol（编辑。） Euro-Par2007，Lecture Notes in Computer Science，vol。 4641页。 514－525.* 斯普林格-维拉格（2007年）。 DOI10.1007/978-3-540-7466-5_55
9. Jelasity，M.，Montresor，A.，Babaoglu，O.：基于流言的大型动态网络聚合。 计算机系统ACM交易23（3），219-252(2005）。 DOI10.1145/1082469.1082470
10. 卡普，R，辛德尔豪尔，C，申克，S，沃金，B：随机谣言传播。 In：第41届计算机科学基础年度研讨会论文集(FOCS‘00)，pp。 565－574. IEEE计算机学会，华盛顿特区，美国（2000年）。 DOI10.1109/sfcs。 2000.892324
11. Kempe，D.，Dobra，A.，Gehrke，J.：基于流言的聚合信息计算。 In：第44届IEEE计算机科学基础研讨会论文集(FOCS‘03)，pp。 482－491. IEEE计算机学会（2003）。 DOI10.1109/sfcs.2003.1238221
12. Kempe，D.，Kleinberg，J.，Demers，A.：空间八卦和资源定位协议。 j。 ACM 51（6），943－967(2004）。 DOI10.1145/1039488.1039491
13. Kempe，D.，Mc Sherry，F.：一种用于光谱分析的分散算法。 In：STOC‘04：第三十六届年度ACM计算理论研讨会论文集，pp。 561－568.ACM，纽约，纽约，美国（2004年）。 DOI10.1145/1007352.1007438
14. 凯尔马雷奇，上午，范斯坦，上午。 (编辑。)：ACM SIGOPS操作系统审查41（2007年）。 关于八卦网络的特刊
15. 金梅尔，A.J.：谣言和谣言控制：经理的理解和打击谣言指南。 Lawrence Erlbaum Associates（2003年）
16. 卢尔，S：谷歌和I.B.M。 加入“云计算”研究。 《纽约时报》（2008年）
17. 皮特：散布谣言。 SIAM应用数学杂志47（1），213-223(1987）。 DOI10.1137/0147013
18. van Renesse，R.，Birman，K.P.，Vogels，W.：Astrolabe：一种用于分布式系统监测、管理和数据挖掘的健壮和可伸缩的技术。 计算机系统ACM交易21（2），164-206(2003）。 DOI10.1145/762483.762485
19. 肖，L，博伊德，S，Lall，S：一种基于平均共识的鲁棒分布式传感器融合方案。 In：IPSN’05：第四届传感器网络信息处理国际研讨会论文集，第9页。 IEEE出版社，Piscataway，新泽西州，美国（2005年）。 DOI10.1109/ipsn.2005.1440896

解决办法

第一章存在的问题

由于这里的问题应该让读者思考（而不是演习），我们只能给出提示，而不是完整的解决方案。

1.1 让我们假设节点是合作的。 如果在每一跳中，八卦消息包含发送方的本地时间，那么发送方和目标的本地时间差异就可以处理，只要传输时间接近于零（就像在无线网络中），或者至少是小的和有界的。 否则，如果一个传输延迟的统计模型是可用的，那么一个人仍然可以有一个相当好的估计，因为估计误差平均超过几个跳，只要模型是无偏的。 但有时传输时间不能近似或模拟一个先验。 在这种情况下，人们可以在飞机上学习这种模型，使用几种技巧，例如注入特殊消息、计数消息的跳数和观察它们到达发送方的时间等等。 我们需要有创造力。

1.2 它有很大的不同。 优化的可能性是无穷无尽的，这种情况类似于一些文件共享系统，在这些系统中，节点确实知道浮动的文件块，并且它们应该拥有这些文件块。 因此，一个人可以应用策略来平衡每个同时消息的传播(就像BitTorrent最罕见的第一策略)，当一个给定的消息应该在很久以前到达时（根据预测的传播时间），一个人可以尝试变得更有侵略性，等等。 此外，基于拉的策略在这里当然更有效，因为如果我们知道周围没有更多的更新，我们可以停止拉。

1.3 异步传播可以更快，因为“幸运”消息可以在到达节点时立即移动到下一跳。 随着概率的越来越小，这种现象可以在很短的时间内连续地重复，一些消息实际上可以覆盖几个跳。 在基于周期的模型中，这是不可能的。这种情况类似于矩阵计算[6]中Jacobi和Gauss-Seidel迭代之间的差异。

1.4 可能会出现问题，例如，如果节点发送推送消息，然后它在收到自己推送消息的答案之前收到推送消息。 如果试图使一对节点之间的值交换（即发送推送消息和接收答案）成为原子事务，那么就会出现死锁方面的高度非平凡的问题，更不用说协议的复杂性了。 只推平均是延迟问题的一个优雅的解决方案。