


技术视角 实现数据复制的一致性

Philip A. Bernstein

如需阅读对应的论文，请访问
/10.1145/2632792。 

云计算系统中的**数据**应该具有高可用性。换言之，无论你何时接入系统，你在系统中保存的数据都应随时可用。实现该功能的标准机制——**数据复制**——需要维护每个用户数据的多个副本。

复制数据本身并不能解决问题，因为可用的数据副本可能是过期的。具体的解释如下，假设系统维护了文件 x 的三个副本： x_1 ， x_2 ，和 x_3 。假设 x_1 和 x_2 当前可用，而 x_3 不可用。如果程序更新 x ，则系统会把新值写入 x_1 和 x_2 ，但由于 x_3 不可用，所以无法更新 x_3 。这不会引发问题，因为系统仍然有两个正确的副本。接下来，假设 x_1 和 x_2 发生故障，然后 x_3 得以恢复。现在，问题出现了。如果用户读取 x ，系统能提供的最佳结果是返回 x_3 的值。但是该副本是过期的——它没有包含最近的更新值。

针对这一问题，早期的解决方案为多数一致。² 处理 x 的写入操作时，系统会分配一个单调递增的时间戳，然后在 x 的多数副本中写入 x 的值和时间戳。处理 x 的读取操作时，系统会读取 x 的多数副本，然后返回具有最大时间戳的值。由于任意两个多数副本的交集至少会包含一个副本，所以每次读操作能够返回之前在该数据上写入的值。

在上文的例子中，读取操作只读取了一个副本，并非三个副本中的多数。因此，可能无法返回 x 的最新状态。如果使用了多数一致，系统就不得不等待第二个副本变为可用，这样才能读够两个副本。接下来，它会返回在读到的两个副本

给定 R 、 W 和 N ，下列论文的作者提出了一个公式用于计算读到的数据不属于最近 K 次写入值之一的概率。

中更接近当前时间的副本，因为该副本的值必定是最新的。

Gifford¹ 把多数的概念扩展为加权多数，并将其称之为**仲裁**。每个副本都会被赋予权重。读取操作必须访问达到读取**仲裁**的副本——副本集合的权重至少达到 R 。而写操作则必须更新达到写入**仲裁**的副本——副本集合的权重至少达到 W 。设定 $R+W$ 必须超过所有副本的总权重 N 后，我们可以确保每次读取均能读到在最近一次写入操作中写入的值。

例如，设系统保存了 x 的四个副本，即 x_1 ~ x_4 。我们可以把 x_1 和 x_2 的权重设为 1；若 x_3 和 x_4 位于更可靠的服务器中，则把 x_3 和 x_4 的权重设为 2。这样得到 $N=6$ 。如果读取比写入更频繁，那么我们可以设 $R=3$ ， $W=4$ ，这样读取执行的任务会比写入少。因为 $R+W>N$ ，所以每次读取均能读到之前在该数据上写入的值。

按理说，每次写入操作都必须更新数据的所有可用副本。但是，读者也必须读多个副本，这颇为麻烦，因为它增加了开销和延迟。如果写入操作经常执行很快，这会特别棘手，因为在这种情况下，即使

每次读取操作没有读取仲裁，也极有可能读到最新的副本。如果读到过期数据的概率足够低，那么让读者读取少于仲裁数量的副本或许是可取的。实际上，某些云应用也正是在这么做的。

直到最近，这种通过猜测实现的折中方法才有所改观。原来对于读到的数据的过期性或承受某些过期风险后期望获得的延迟减少缺乏界定。后面的论文阐述了一项突破性进展，它抛开了猜测，用名为**概率有界过期性**的原理分析取而代之。给定 R 、 W 和 N ，论文的作者提出了一个公式用于计算读到的数据不属于最近 K 次写入值之一的概率。为了计算读到的数据不是在最近 Δ 时间单位内写入的值的概率，他们使用了基于时间参数的蒙特卡罗模拟，且这些参数可以在运行的系统中通过测量得出。不仅如此，他们基于开源记录管理器实现了该机制，让这种分析变得切实可行，使得用户可以依据自己的判断对过期性和延迟进行权衡。而原先的猜测工作现在也变成了人们可以精心策划的目标。

参考资料

1. Gifford, D.K. Weighted voting for replicated data. *SOSP* (1979), 150–162.
2. Thomas, R.H. A majority consensus approach to concurrency control for multiple-copy databases. *ACM Trans. Database Syst.* 4, 2 (1979), 180–209.

Philip A. Bernstein (<http://research.microsoft.com/~philbe>) 是华盛顿州雷德蒙市微软研究院的杰出科学家。

译文责任编辑：陈海波

版权归属于作者 / 所有者。

Technical Perspective

Getting Consensus for Data Replication

By Philip A. Bernstein

DATA IN A cloud computing system should be highly available. That is, whenever you connect to the system, the data you stored there should be ready to use. The standard mechanism to accomplish this—*data replication*—involves maintaining multiple copies of each user's data.

By itself, replicating data does not solve the problem, because the copy of data that is available might be stale. To see why, consider a system that maintains three copies of file x : x_1 , x_2 , and x_3 . Suppose x_1 and x_2 are currently available and x_3 is down. If a program updates x , the system writes the update to x_1 and x_2 , but not to x_3 because it is down. No problem, since the system still has two correct copies. Next, suppose x_1 and x_2 fail, and then x_3 recovers. Now there is a problem. If a user reads x , the best the system can do is return the value of x_3 . But that copy is stale—it does not have the latest update.

An early solution to this problem is called *majority consensus*.² To process a write operation on x , the system assigns a monotonically increasing timestamp to the write, and writes x 's value and timestamp to a majority of the copies of x . To process a read operation on x , it reads a majority of the copies of x and returns one that has the largest timestamp. Since the intersection of any two majorities includes at least one copy, each read returns the result of the previous writes on the same data.

In our example, the read operation only reads one copy, which is not a majority of three. Therefore, it might not return the latest state of x . Using majority consensus, the system would have to wait until a second copy became available, so it could read two copies. Then, it could return the more up-to-date copy of the two that it reads, which must be the freshest.

The notion of majority was extended in Gifford¹ to be a weighted major-


Given R , W , and N , the authors of the following paper offer a formula to calculate the probability of reading data that was not written by one of the K most recent writes.

ity, which is called a *quorum*. Each copy is given a weight. A read must access a *read quorum* of copies—a set of copies whose weight is at least R . And a write must update a *write quorum* of copies—a set whose weight is at least W . By requiring that $R+W$ exceeds the total weight N of all copies, we are assured that each read reads a copy that was written by the last write.

For example, suppose the system stores four copies of x , namely, x_1 - x_4 . We might assign a weight of 1 to x_1 and x_2 , but assign a weight of 2 to x_3 and x_4 if they are stored on more reliable servers. So $N=6$. If reads are more frequent than writes, we might set $R=3$ and $W=4$, so that reads have less work to do than writes. Since $R+W>N$, each read operation reads the result of the previous write on the same data.

It stands to reason that each write has to update all available copies of the data. But it is annoying that readers have to read multiple copies too, since it adds overhead and delay. It is especially annoying if writes usually execute quickly, since in this case each read is very likely to read the latest copy even if it does not read a

quorum. If the probability of reading stale data is sufficiently small, then it might be satisfactory to allow readers to read less than a quorum of copies. In fact, some cloud applications do exactly that.

Until recently, this compromise was done by guesswork, without a bound on the staleness of the data that is read or the expected latency improvement from risking some staleness. The following paper is a breakthrough that removes the guesswork and replaces it by a principled analysis, called probabilistically bounded staleness. Given R , W , and N , the authors offer a formula to calculate the probability of reading data that was not written by one of the K most recent writes. To calculate the probability of reading data that was not written in the last delta time units, they use a Monte Carlo simulation based on time parameters that can be measured in a running system. Moreover, they made the analysis practical by implementing it in open source record managers, so that users can make their own judgment on trading off staleness for latency. What was formerly a guess is now a goal you can engineer for. 

References

1. Gifford, D.K. Weighted voting for replicated data. *SOSP* (1979), 150–162.
2. Thomas, R.H. A majority consensus approach to concurrency control for multiple-copy databases. *ACM Trans. Database Syst.* 4, 2 (1979), 180–209.

Philip A. Bernstein (<http://research.microsoft.com/~philbe>) is a Distinguished Scientist at Microsoft Research, Redmond, WA.

利用 PBS 量化最终一致性

Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, Ion Stoica

摘要

数据复制造成人们需要在操作延迟和一致性之间做出根本性的权衡。在可用的一致性模型范围中较弱的一端是最终一致性，它对返回数据的过期性没有施加限制。不过，说来有趣，对于从业者而言，最终一致性在延迟和可用性方面的收益往往“足够好”。在本文中，我们解释了这一现象，并论证了下列观点：虽然实现最终一致性的系统只能提供较弱的保证，但是他们通常能够返回一致的数据。不仅如此，与实现强一致性的同类系统相比，它们的延迟更低。为了量化实现最终一致性的数据存储的行为，我们引入了概率有界过期性（PBS）。这种模型从版本和墙上时钟时间两方面为数据过期性提供了预期边界。我们为基于版本的过期性推导出了一个闭式解。而且，对于一大类采用仲裁复制的 Dynamo 风格存储系统，我们构建了实时过期性模型。使用 PBS 后，在互联网生产级负载下，我们对部分的、非重叠的仲裁系统测量了在延迟和一致性之间做出的权衡。我们采用量化的方式阐述了最终一致性系统呈现下列特点的实现方式和原因：在提供相当大的延迟收益的同时，它们往往能在几十毫秒的时间内返回一致的数据。

1. 概述

现代的分布式数据存储需要变得可伸缩、高可用且访问速度快。基于至少以下的两点原因，这些系统通常会把数据复制到不同的机器上，并且会越来越多地把数据复制到不同的数据中心上：首先，在组件出现故障时提供可用性；其次，通过使用多个副本来响应请求，提供更好的性能。配置和维护复制数据对应用和数据存储设计产生了重要影响。¹ 对于一大类的应用而言，大规模集群的性能举足轻重。在实践中，增加延迟往往会导致收入大幅减少。²² 例如，在亚马逊公司，100ms 的额外延迟会导致销售额下降 1%，¹⁵ 而进行谷歌搜索时如果出现 500ms 的额外延迟，则会导致对应的流量下降 20%。¹⁶ 然而，降低分布式数据存储的延迟需要付出一定代价：如每次操作访问更少的副本，则会对可实现的语义保证造成不利影响。

为了提供可预测的低延迟，现代系统通常会避开保证读操作“强”一致性的协议（例如，单一副本假

象），而倾向于选择通常以最终一致性呈现的“较弱”的语义。^{1,5,7,10} 这种最终一致性是现代存储系统提供的最弱的属性之一：在无新的写入操作的情况下，读取操作最终会返回最近的（单次或多次）写入操作的结果；除了这点之外，它对数据过期性不提供任何保证。²⁶ 根据这一定义，只要在未来的某个时间点上，该存储返回了最后写入的数据⁴，那么返回时间有数周之久的数据的存储就是最终一致的，返回任意数据的存储也是一致的（比如，返回值一直是 42）。对于终端用户而言，由于几乎没有有用的语义，使用最终一致性的决策往往饱受争议。^{12,23,24} 在现在提供最终一致性的很多生产存储中^{10,14}，用户几乎无法洞悉其存储的行为或其数据的一致性。在复制配置多变的情况下，情况更为明显。不过，最终一致性系统的广泛部署暗示，应用往往可以容忍偶发的过期性，而且在很多情况下，数据更倾向于“足够新”。

在本文中，我们通过量化最终一致性在最终性和（非）一致性方面所达到的程度并解释原因，从而弥合了理论保证与当前实践之间的鸿沟。事实上，在最坏的情况下，最终一致性会导致数据过期性的程度没有任何边界。然而，接下来我们会说明，通常的情况往往有所不同。我们论文的核心为下列观测结果，给定特定的工作负载和部署环境后，可以为最终一致性构建模型，用于提供一致性的概率期望。因此，对于程度不同的确定性而言，依照其可能偏离强一致行为的程度，最终一致的存储系统可对此提供界定。我们陈述了用于此类边界的概率模型，称之为概率有界过期性，或 PBS。

为了利用 PBS 预测一致性，我们需要知道最终一致的存储返回过期数据的时间和原因，并了解如何量化其返回数据的过期性。在本文中，我们提出了一些算法和模型用于测量文献中常见的两种过期

本论文旧版的标题为《用于实用的部分仲裁系统的概率有界过期性》，曾在 VLDB 2012 中发表。⁵ 本文受邀扩展版本的标题为《使用 PBS 量化最终一致性》，将会在 2014 年 VLDB Journal 的《VLDB 2012 最佳论文》中发表。⁷ 本文的部分内容还发表在 SIGMOD 2013 的演示中，题为《PBS 的运用：使用一致性度量提升数据管理》。⁶

性度量指标：墙上时钟时间²¹和版本²⁷。**PBS**描述了这两种度量指标，提供了在写操作返回 $((\Delta, p)$ -语义或是“什么程度的最终才算最终一致性？”) Δ 秒后，读到该写入值的概率，读到数据项 $((K, p)$ -语义，或是“什么程度的一致才算最终一致性？”)的最近 K 个版本之一的概率，以及经历两者结合 $((K, \Delta, p)$ -语义)后的概率。**PBS**并未提出新的机制来加强确定性的过期性边界；²⁷与此相反，我们的目标是提供一个新的视角来分析、改进和预测广泛部署的现有系统的行为。

在本文中，我们把**PBS**应用于仲裁复制的数据存储中，比如Dynamo¹⁰以及基于Dynamo产生的几个开源系统。通过确保读取操作和写入操作中的副本集存在重合，仲裁系统确保了副本读写操作的强一致性。不过，利用部分（或非严格的）仲裁后，由于需要做出响应的副本得以减少，延迟得以降低。利用部分仲裁后，写入和读取的副本集不需要重合：给定 N 个副本，设读取仲裁和写入仲裁大小为 R 和 W ，部分仲裁意味着 $R + W \leq N$ 。对于部分仲裁，我们推导出了用于**PBS** $((K, p)$ -正规语义的闭式解，并使用了蒙特卡罗方法探索延迟与 $((\Delta, p)$ -正规语义之间的权衡。

最后，对于在生产中部署的Dynamo风格的数据存储系统，我们使用**PBS**研究了其在正常情况下（即无故障场景中）观测到的过期性。我们说明了写入延迟的高方差会如何导致更宽的窗口。例如，在某个生产环境中，从旋转硬盘转换到固态硬盘后，由于写入延迟的均值和方差减少，预期一致性有了大幅提升（例如，要达到99.9%的一致性读取概率，等待时间为1.85 ms，而不是45.5 ms）。我们还对部分仲裁提供在延迟与一致性上的权衡做了定量观测。例如，在另一个生产环境中，**PBS**的计算说明，对于202ms的不一致窗口而言，在第99.9个百分位处（230至43.3ms）出现了81.1%的复合读取和写入延迟提升（一致性读取的概率为99.9%）。该分析帮助业界论证了性能收益，可促使运营商选择最终一致性；它还推动了其他终端用户应用的发展，应用的范围涵盖了从一致性监测到基于一致性的服务水平协议（SLA）和查询规划等众多领域。

2. 概率有界过期性

在通常定义中，最终一致的数据存储系统并未对返回数据的新近程度做出任何保证：“如果对象没有得到任何新的更新，那么最终所有的访问均会返回最后更新的值。”²⁶活跃度是一个有用的属性，它保证某个好的东西最终会发生，但是它并不提供任何安全属性：在中间阶段，数据存储可能返回任何数据。“4很多真实世界中的最终一致性存储并未做出超过上述定义的任何保证，但是它们仍然得到广泛部署。在过去的十年中，它们的流行程度与日俱增（见第3.2节）。接

下来我们会看到，大多数最终一致性存储系统使用的协议确实没有强加额外的保证，但在运行中它们可能提供了这些保证。

为了量化没有保证但又往往提供的语义，我们开发了用于推断一致性的概率框架，将其称为概率有界过期性，或**PBS**。数据存储可选择提供的一致性模型范围广泛，从线性一致性，到因果一致性，再到最终一致性；如果给定的一致性模型没有得到保证，终端用户观测到该模型的可能性有多大呢？本文中我们研究了以过期性的形式呈现的两种经典（非）一致性的变体：版本和时间。我们开发了不同的**PBS**度量标准，用于对下列情况提供量化期望：存储系统返回最近 K 次写入（若 $K=1$ ，则为最近的）的版本之一；以及存储系统返回 Δ 秒之前的最新版本。最后，虽然**PBS**并不提供保证，但它仍然有助于推断和反思给定系统的行为，这点与用于衡量性能的现代SLA类似（见第6节）。

首先，我们需要为“强一致性”语义定义一个基线。我们研究的Dynamo风格存储系统允许我们在“强”正规语义和最终一致性之间进行选择；然后，我们观测在什么时候所选的最终一致性与对应的“强”语义的行为类似。在分布式系统的文献中：²

定义1：若满足下列条件，则说明给定数据项的读取操作符合正规语义：如果某数据项的读取操作与写入操作没有重合（实际时间），则返回最后完成的写入结果；或者，如果某数据项的读取操作与至少一次写入操作重合（实际时间），则返回最后完成的写入结果，或是重合的某次写入操作的最终结果。

据此，正规语义提供了一种假象，让人们觉得各复制数据项只有一个副本。但是，在出现并发的读写操作时，写入操作的值可能变为可见但随后“消失”。尽管这种特别的、重合的情况有点让人难以摸透（参考线性一致性的定义¹³），但这种数据复制配置仍被广泛部署。

在我们把**PBS**应用于正规语义前，我们先概括下用于解释多版本过期性的语义：²

定义2：若满足下列条件，则说明给定数据项的读取操作符合 K -正规语义：如果某数据项的读取操作与写入操作没有重合（实际时间），则返回的结果为最近 K 次已完成的写入值之一；或者，如果某数据项的读取操作与至少一次写入操作重合（实际时间），则返回的结果为最近 K 次已完成的写入值之一，或是重合的某次写入操作的最终结果。

K -正规语义可用于推断给定读取操作的过期程度，也可用于强加额外的一致性属性，比如单调读，即读操作的结果不会出现“回到过去”。^{5,25}

现在，我们来介绍**PBS**原理的首个应用：给定不保证 K -正规语义的系统，我们可以采用概率方法来推断它的语义：

定义 3: 若系统中每次读取操作返回 K - 正规语义的概率为 p , 则该系统提供了 (K, p) - 正规语义。

这种修改简单又直接: 给定已有的语义保证, 我们可以考虑其概率版本, 其中读取有可能会也可能不会遵守属性的规定。当然, 上面只是一个定义, 我们仍然必须确定如何实际提供 PBS 预测, 这也是下文的重点。

除了考虑基于版本的过期性外, 我们还可以考虑实际时间方面的过期性。我们会看到, 消息传播和处理延迟能够影响跨时间的一致性, 所以我们把正规语义进行了扩展, 纳入了时间因素:

定义 4: 若读操作的返回结果为最多 Δ 单位时间前的最新写入值, 或是最多 Δ 单位时间前已经开始但尚未完成的写入结果, 则说明读取遵守 Δ - 正规语义。

与上文类似, 我们可以把这个定义扩展到概率场景中:

定义 5: 若系统中每次读取操作遵守 Δ - 正规语义的概率为 p , 则说明该系统提供了 (Δ, p) - 正规语义。

虽然在本文中我们未考虑上述定义, 但是可以同时考虑时间和版本过期性, 我们将其作为术语 (K, Δ, p) - 语义。^{5,7}

3. 仲裁系统的背景

有了 PBS 度量标准后, 我们可以更进一步, 把他们应用于实际系统中, 发挥它们的最大作用。在本文的研究中, 我们探讨了 PBS 度量标准在仲裁 (基于复制的数据存储系统) 中的应用情况; 它们代表了一大类广泛部署的分布式数据存储系统。不过, 基于我们的一些成果, 我们相信该方法也可应用于其他风格的基于复制机制的系统。接下来, 我们会阐述仲裁系统的背景, 并重点关注当前的实践。

3.1. 仲裁的基础: 理论

作为一种分布式数据的复制策略, 仲裁系统有着很长的传统。²⁰ 在基于仲裁的复制机制中, 当数据存储系统写入数据项时, 会把数据项发给负责管理副本的服务器集合, 我们将其称之为写入仲裁。为了提供读取操作, 数据存储系统会从可能存在差异的副本集合中获取数据, 我们将这种集合称之为读取仲裁。对读取操作而言, 数据存储系统会使用全序版本信息比较副本返回的值集, 并能够返回最近的值 (或者如果需要, 返回所有接收的值)。对于每个操作而言, 数据存储系统从副本集合的某个集合中选择 (读取或写入) 仲裁, 我们将其称之为仲裁系统, 其中每个数据项拥有一个仲裁系统。仲裁系统有很多种类型, 但是, 对于大小为 N 的副本集而言, 有一种简单的配置是使用固定大小的读取和写入仲裁, 我们将其标记为 R 和 W 。严格的仲裁系统有着仲裁系统中任何两个仲裁不重合 (空交集) 的性质, 提供了正规语义。严格仲裁系统的一个简单范例是多数仲裁系统, 其使用副本的数量作为

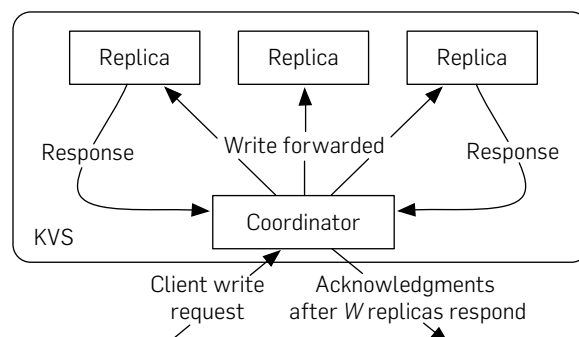
仲裁 $\lceil \frac{N+1}{2} \rceil$ 。本文中将要研究部分仲裁系统, 放宽了严格仲裁系统中约束: 即在部分仲裁系统中, 至少有两个仲裁间没有重合副本。¹⁹

3.2. 仲裁的基础: 实践

在实践中, 很多分布式数据管理系统利用仲裁作为复制机制。亚马逊公司的 Dynamo¹⁰ 作为先驱引领了一类基于最终一致的数据存储系统, 其中包括 Apache Cassandra^a、Basho Riak^b 和 Project Voldemort^c。所有这些系统都使用了仲裁风格复制机制的相同变体, 而且据我们所知, 在广泛采用的数据存储中, 各种存储使用的基于仲裁的复制协议不存在本质区别。

Dynamo 风格的仲裁系统为每个数据项设置了一个仲裁系统; 通常情况下, 会使用一致性哈希方案或集中式的成员协议对数据项在仲裁系统进行映射。系统集群中的每台服务器均保存有多个数据项。如图 1 所示, 客户端向系统集群中的服务器发出读取和写入的请求, 然后服务器会把该请求转发给操作涉及的数据项的所有副本。当协调服务器从预定数量的副本中收到响应后 (通常基于每个操作进行配置), 该服务器认为操作已经完成。因此, 不会有消息丢失, 所有的副本最终会收到所有的写入值。Dynamo 把数据项的复制因子设为 N , 成功读取需要的副本响应数量为 R , 成功写入需要的副本确认数量为 W 。与其他严格的仲裁系统类似, 在无故障操作期间, 若 $R + W > N$, 则 Dynamo 提供了正规语义。不过, 与传统的仲裁系统有所不同, 即使在操作返回后, Dynamo 的写入仲裁大小也会增加, 其会通过反熵增大。^{4,10} 协调器会把所有的请求发给所有的副本, 但是只考虑最先的 R (W) 个响应。基于命名方面的考虑 (为了与“动态”仲裁成员协议加以区分), 我们把这些系统称为扩展型部分仲裁系统。

图 1 客户端向 Dynamo 风格的仲裁进行写入时的控制流示意图 ($N = 3, W = 2$)。协调器服务器处理客户端的写入操作, 把它发给所有 N 个副本。当协调器收到 W 次确认后, 写入调用返回。



^a <http://cassandra.apache.org/>

^b <http://www.basho.com/riak/>

^c <http://www.project-voldemort.com/>

如同我们在本论文的扩展版本中的讨论一样,^{5,7}, 系统操作员往往会报告他们在 **Dynamo** 风格的存储中使用了部分仲裁配置, 并提到在“通常情况”下达到了“最高性能”, 特别是对于“低价值”的数据或“延迟极低且可用性高”的查询而言。

4. PBS 和部分仲裁

既然我们已经有了 **PBS** 度量标准, 也了解了仲裁的行为, 接下来我们可以构建多种模型分析部分仲裁中的一致性概率。在本文中, 我们简要讨论了传统概率仲裁使用的基于版本的过期性, 并开发了一个更为复杂的、基于时间的过期性的“白盒”模型, 用于分析 **Dynamo** 风格系统的基于时间的过期性。

4.1. PBS (K, p) - 正规语义

为了理解静态的、非扩展的仲裁行为, 我们首先重温下概率仲裁系统¹⁹。在部分仲裁系统中, 它提供了仲裁交集的概率保证。举例而言, 考虑一下 N 个副本的情况, 其中读取和写入仲裁的大小分别为通过均匀随机抽取的 R 和 W 。我们能够计算出读取仲裁未包含最后写入版本的概率。该概率的大小为, 由写入仲裁中未写入值的副本组成的, 大小为 R 的仲裁的数量除以可能的读取仲裁数量:

$$p_s = \frac{\binom{N-W}{R}}{\binom{N}{R}} \quad 1.$$

如果 N 的值小, 不一致性的概率会相当高。然而, 通过缩放副本的数量和仲裁的大小, 人们可以获得任意数值的高一致性概率。¹⁹ 例如, 设 $N=3, R=W=1$, $p_s=0.6$, 但设 $N=100, R=W=30$, $p_s=1.88 \times 10^{-6}$ 。² 这让人联想到了生日悖论: 随着副本数量的增加, 在任何两个仲裁之间不出现交集的概率减少。因此, 这些系统的渐进性相当好——但是也只在渐进性的尺度上如此。

虽然概率仲裁让我们可以确定返回值为最近写入数据库的值的概率, 但是它们却无法描述在未返回最近值时发生的情况。在本文中, 我们确定了返回值属于限定数量的版本之一的概率 ((K, p) - 正规语义)。在下面的阐释中, 我们考虑了传统的、非扩展的写入仲裁 (无反熵)。

与前面的示例类似, 选定独立的、恒等分布 (IID) 的读取和写入仲裁后, 返回值为最后 k 次写入版本之一等同于让 k 个独立的写入仲裁相交。假定单一仲裁不交的概率为 p , 则其与最近 k 个独立仲裁之一不交的概率为 p^k 。因此, 在我们的示例仲裁系统中, 与均匀随机抽取的选择性不交的概率为等式 1 的 k 次方。

$$p = \frac{\binom{N-W}{R}}{\binom{N}{R}} \quad 2.$$

若 $N=3, R=W=1$, 这意味着返回值为 2 个版本之一的概率为 0.5; 返回值为 3 个版本之一的概率为 0.703; 5 个版本, >0.868; 10 个版本, >0.98。若 $N=3, R=1, W=2$ (或者, 等价的 $R=2, W=1$), 这些概率增大: $k=1 \rightarrow 0.6, k=2 \rightarrow 0.8$, 以及 $k=5 \rightarrow >0.995$ 。

该闭式解对大小不随时间改变的仲裁成立。对于扩展的部分仲裁系统, 该解为 (K, p) - 正规语义的 p 的下界。在本论文的完整版本中, 我们对本分析进行了更为深入的阐释, 包括单调读的一致性分析, 仲裁负载, 以及时间和版本混合的闭式解。^{5,7}

4.2. Dynamo 中的一致性

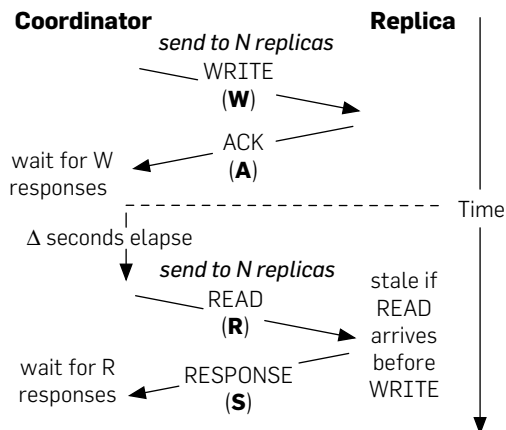
我们有一个简单的闭式模型用于 (K, p) - 正规语义, 但是基于时间 ((Δ, p) - 正规) 语义取决于给定系统采用的仲裁复制算法、工作负载和任意的反熵流程。在本节中, 我们开发了多种技术在 **Dynamo** 风格的数据存储系统中分析 **PBS** (Δ, p) - 正规语义。

由于读写消息重排序及其造成的消息延迟等原因, **Dynamo** 风格的仲裁系统是不一致的。在本文中, 我们采用了白盒方法分析不一致性, 并直接检查一致性背后的协议。相应地, 我们开发了 **Dynamo** 操作的消息延迟模型, 用于捕捉写入请求 (w)、写入确认 (a)、读取请求 (r) 和读取响应 (s) 的消息延迟的影响。为方便起见, 我们把它们统称为 **WARS**。在图 2 中, 我们使用消息的时空图说明了 **WARS**。图中说明了在写入操作完成 Δ 秒后再进行读取操作的过程中, 协调器和单一副本之间的消息传递情况。相应地, 此处的 Δ 对应 **PBS** (Δ, p) - 正规语义中的 Δ 。简言之, 在最后 (已完成) 的写入请求到达前, 如果所有最先的 R 均响应了到达它们的副本的读取请求, 则读到的值是过期的。

对于写入操作而言, 协调器会发出 N 条消息, 每个副本一条消息。从协调器向副本发出的、包含写入操作的消息延迟程度为从分布中推导出的值 w 。协调器等待从副本中发出的 W 次响应, 然后才认为写入操作完成。确认写入操作的每条响应的延迟程度为从分布中推导出的值 a 。

对于读取操作而言, 协调器 (可能与写入操作的协调器不同, 且代表的客户端与发出写入操作的客户端也可能有所不同) 发出 N 条消息, 每个副本一条消息。从协调器向副本发出的、包含读取请求的消息延迟程度为从分布中推导出的值 r 。协调器等待从副本中发出的 R 次响应, 然后才返回它接收到的最新的值。

图 2 通过为消息延迟（执行写入操作 t 秒后再执行读取操作时，协调器和副本之间传递的消息）建模，WARS 模型描述了 Dynamo 中的过期性。在拥有 N 个副本的系统中，图中描绘的消息与 N 副本进行交流。



每个副本响应读取操作的延迟程度为从分布中推导出的值 s 。

如果读取操作的协调器最先接受到的 R 次响应在副本接受最新的版本前到达副本（延迟程度为 w ），则读取协调器会返回过期数据。如果 $R + W > N$ ，这不可能发生。然而，在部分仲裁中，上述情况的发生频率取决于延迟的分布情况。如果我们把写入操作的完成时间（若协调器已经收到 W 次确认）标为 w_t ，对于从 R 中推导出的 r' 以及从 W 中推导出的 w' ，若 $r' + w_t + \Delta < w'$ ，则单一副本的响应是过期的。在协调器等待所有所需的确认（A）和副本等待读取请求（R）时，写入操作有时间传播到更多的副本。在传回（S）读取操作的协调器的过程中，读取操作的响应又被延迟了，造成重排序的可能性增大。定性地来看，由于重排序的原因，呈长尾分布的写入操作（W）和相对较快的读取操作（R, S）会增大过期性的几率。

WARS 考虑了消息发送、延迟和接收的影响，但也提出了一个难以进行确切分析的问题，因为它带有几个非独立阶统计量。正如我们在第 5.1 节中讨论的那样，我们会使用蒙特卡罗方法探讨 WARS，因为理解和实现该方法相当简单。我们已经发现，给定真实世界系统的行为轨迹后，对 WARS 分布进行参数化相当容易（见第 5.3 节）。

5. PBS 实战

既然我们已经拥有了 Dynamo 风格存储的 PBS 模型，现在我们就把它们应用到真实世界系统中去。如同第 4.2 节中讨论的那样，给定系统中的 PBS (Δ, p) - 正规语义取决于读取操作和写入操作在副本间的传播。我们引入了 WARS 作为一种手段来推断 Dynamo 风格的仲裁系统中的不一致性，但在实践中观测到的定量度量指标（如过期性）取决于 WARS 的各种延迟分布

情况。在本节中，我们依照人工创建的分布和真实世界的分布对 Dynamo 风格 (Δ, p) - 正规语义进行了分析，以便更好地理解“最终一致”意味着“一致”的频率有多高。更重要的是，还能让人更好地理解造成 Dynamo 风格的存储确实经常一致的原因。

部分仲裁的 PBS (K, p) - 正规分析可以较容易地以闭式的形式获取（见第 4.1 节）。它不依赖于写入操作的延迟或任何环境变量。实际上，在实践中如果没有扩展仲裁或反熵，我们观测到我们推导出的方程在实验中成立。

相比之下， (Δ, p) - 正规语义依赖于反熵，所以要复杂得多。在本节中，我们的重点放在 PBS (Δ, p) - 正规语义的实验期望的推导上。首先，我们基于 WARS 模型验证了我们的蒙特卡罗分析，其中使用了从伯克利的 Cassandra 集群中收集到的消息级轨迹。然后，我们探讨了人工创建的延迟分布。在剩余的分析中，我们探讨了源于下列两家互联网公司的分布：领英（LinkedIn）和 Yammer。

5.1. 蒙特卡罗模拟

我们在基于蒙特卡罗的模拟中实现了 WARS 分析。给定 Δ 的值后，计算 (Δ, p) - 正规语义简单又直接（见 Bailis 等人的伪码⁷）。为了模拟协调器与 N 个副本中每个副本之间的消息延迟，我们把从分布 D 中取出的第 i 个样本标记为 $D[i]$ ，并从 W, A, R 和 S 中取出 N 个样本。然后计算写入请求完成的时间（ w_t ，或是协调器得到其第 W 个确认的时间； $\{W[i] + A[i], i \in [0, N]\}$ 的第 W 个最小值）。接下来，确定在最先的 R 个副本中，是否有任何副本包含最新的响应：检查 R 中最先的 R 个样本，依据 $R[i] + S[i]$ 排序，须符合 $w_t + R[i] + \Delta \leq W[i]$ 的要求。多次重复这一过程后，我们可以获得对由轨迹确定的行为的近似。给定写入操作的到达时间分布，如果要把上述公式拓展到 (K, Δ, p) - 正规语义的分析中去，则需要考虑跨时间的多次写入操作。如同本论文的扩展版本中描述的那样，^{5,7} 我们利用从真实世界的 Cassandra 集群中收集到的轨迹验证了上述分析。我们观察到， (Δ, p) - 正规语义的预测出现了 0.28% 的均方根误差，延迟预测出现了 0.48% 的平均标准均方根误差。

5.2. 写操作延迟分布的影响

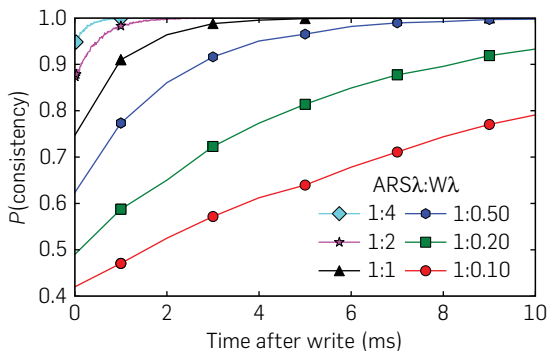
Dynamo 风格系统的 WARS 模型说明，延迟的高方差增大了过期性。在研究真实世界的工作负载前（第 5.3 节），我们用人工创建的分布来单独量化该行为：我们快速观察了很多呈指数分布的写入操作（改变参数 λ ，因为它决定了分布的均值和尾），同时固定 $A = R = S$ 。

我们的结果说明了这种关系，见图 3。当 W 的方差和均值为 0.0625 ms 和 0.25 ms 时（ $\lambda = 4$ ， $A = R = S = 1$ ms 时均值的四分之一），我们观察到，在紧接

着写操作后的一致性几率为 94%；而间隔 1ms 后的几率为 99.9%。然而，当 w 的方差和均值为 100ms 和 10ms 时 ($\lambda = 0.1$, $A = R = S = 1$ ms 时均值的 10 倍)，我们观察到，在紧接写着操作后的一致性几率为 41%；只有在间隔 65ms 后的几率才为 99.9%。随着方差和均值增大，不一致性的概率也会增大。分布的均值固定但方差可变（均匀分布，正态分布）时，我们观察到，若 w 严格大于 $A = R = S$ ，则 w 的均值没有它的方差重要。

减少 w 的均值和方差会提高一致性读取的概率。这意味着，降低写入延迟的方差的技术可以导致更一致的读取操作，接下来我们将会看到这一点。与增加读取和写入仲裁的大小相反，操作员可以选择通过硬件配置或通过推迟读取操作降低（相对的） w 延迟。但是，对于读取操作占主要地位的工作负载而言，后者会损害性能并可能造成不良的排队效应。尽管如此，PBS 分析还是说明了一个事实，即除了简单地调整仲裁大小外，还有多种途径可以避免读取过期的值。

图 3 (Δ, p)- 正规语义，其中 w 的延迟分布呈指数增长且 $A = R = S$ 。平均延迟为 $1/\lambda$ 。 $N = 3, R = W = 1$ 。



5.3. 生产中延迟的分布

为了研究真实世界的行为，我们从两个互联网规模的公司获取了生产中的延迟统计数据。虽然消息级的 **WARS** 时间轨迹可以提供更为准确的预测，但是我们还选择了从务实的角度进行折中分析：正如我们在本文的扩展版本中描述的那样，我们让 **WARS** 分布变得适于分析所获取的各种统计数据（图 4）。^{5,7}

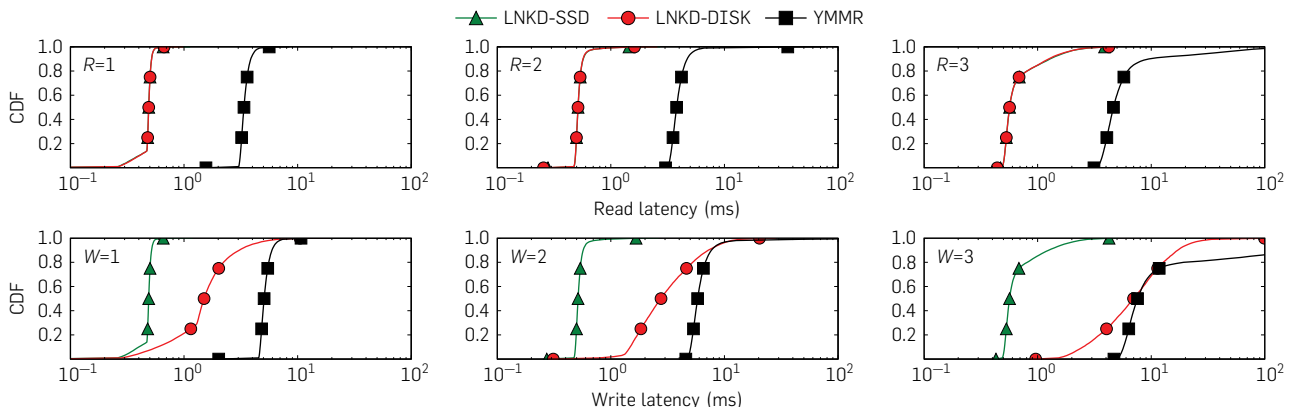
领英 (**LinkedIn**^d) 是一家专业性的在线社交网络。至 2013 年 7 月止，它拥有超过 2 亿 2 千 5 百万会员。为了提供高可用、低延时的数据存储，**LinkedIn** 的工程师搭建了 **Voldemort**。Alex Feinberg 是 **Voldemort** 的首席工程师。他慷慨地为我们提供了在处理 **LinkedIn** 中面向用户的服务时，单台服务器在高峰流量下的延迟分布数据，其中呈现了 60% 的读和 40% 的读 - 改 - 写流量。^{5,7} Feinberg 报告说，使用旋转硬盘时，**Voldemort** “在很大程度上受 IO 的限制，延迟大部分由我们使用的硬盘类型、数据与内存的比值以及请求分发决定”。使用固态硬盘 (**SSD**) 后，**Voldemort** 往往“受 CPU 和 / 或网络的限制”，而且“最大延迟通常由【垃圾回收】活动决定（垃圾回收极少发生，但偶尔也会发生），且时间在几百毫秒之内。”我们把 **LinkedIn** 使用旋转硬盘时的分布标为 **LNKD-DISK**，**SSD** 的轨迹标为 **LNKD-SSD**。

到 2013 年 7 月止，**Yammer**^e 向 20 多万家公司提供了私有社交网络，它使用了 **Basho** 的 **Riak** 存储某些客户数据。基础设施架构师 Coda Hale 为我们提供了生产环境中的 **Riak** 部署的性能统计数据。^{5,7} Hale 提到，“读取和写入拥有截然不同的预期延迟，对 **Riak** 而言情况更为明显。”**Riak** 会延迟写入，“直到 **fsync** 返回，所以读取通常 <1 ms，而写入极少 <1 ms。”此外，虽然我们没有明确地为其建模，Hale 注意到，值所占的

^d <http://www.linkedin.com/>

^e <http://www.yammer.com/>

图 4 生产中读取和写入操作的延迟，适用于 $N = 3$ 和变化的 R 和 W 。对于读操作，**LNKD-SSD** 与 **LNKD-DISK** 等效。图中绘制的点标出了重要的百分位以便对比。较高的 R 和 W 值会导致延迟增加。



空间大小相当重要; 他声称, “在对值进行 LZF 压缩后, 获得了相当大的性能提升。” 我们把 Yammer 的延迟分布标为 YMMR。

5.4. 生产中的过期性

生产延迟的分布确认了在使用最终一致性的存储中过期性往往会受到限制。我们测量了每个分布的 (Δ, p) -正规语义 (见图 5)。LNKD-SSD 和 LNKD-DISK 的数据说明了实践中写入延迟的重要性。紧接着写入完成后 LNKD-SSD 的一致性读取的概率为 97.4%; 而间隔 5 ms 后的一致性读取的概率超过了 99.999%。在紧接着写入完成后 LNKD-SSD 的读取操作几乎能与写入操作平齐。然而, 在写入操作后的若干毫秒内, 读取操作在最后写入前到达的几率几乎为零。该分布的读取和写入操作延迟很低 (中位数为 0.489 ms), 而且由于该分布是短尾的 (第 99.9 个百分位为 0.657 ms), 所以跨所有副本的写入操作均能很快完成。相比之下, 在 LNKD-DISK 数据中, 写入操作的时间要长很多 (中位数为 1.50 ms), 而且存在较长的尾 (第 99.9 个百分位为 10.47 ms)。LNKD-DISK 的 (Δ, p) -正规语义反应了这一差异: 紧接着写入完成后 LNKD-DISK 的一致性读取的概率只有 43.9%; 而间隔 10 ms 后的

概率只达到了 92.5%。这说明, 由于 SSD 降低了写入操作的方差, 所以可能大大提升了一致性。与此类似, 人们可能会期望, 如果使用明确的内存管理来替代无法计划的垃圾回收, 那么一致性可能会有所提升。

在另一个分布中, 我们经历了类似的行为。紧接着写入操作完成后 ($\Delta = 0$) YMMR 的一致性概率为 $p = 89.3\%$ 。不过, 由于其写入分布存在高方差和长尾现象, 在 $\Delta = 1364$ ms 时, YMMR 分布的一致性的概率仅达到 $p = 99.9\%$ 。这意味着, 给定数据项的多个副本, 同步把值写入磁盘能获得稳定性方面的收益, 但却可能会对一致性产生不利影响。另一种提高一致性并避免高方差的方法是依赖多副本 (内存或缓冲区高速缓存) 复制实现稳定性, 但只用异步的方式写入值。

5.5. 仲裁大小的调整

除了使用 $N=3$ ——即我们在实践中碰到的最常见的仲裁大小外——我们还考虑了, 如果保持 $R=W=1$ 不变, 改变副本的数量 (N) 会对 (Δ, p) -正规语义造成何种影响。图 6 描绘的结果说明了随着 N 的增加, 紧接着写入完成后的一致性概率会随之降低。如果副本的数量为二, 则紧接着写入完成后 LNKD-SSD 的一致性读取的概率为 57.5%; 如果副本的数量为 10, 则概率

图 5 用于生产操作延迟的 (Δ, p) -正规语义

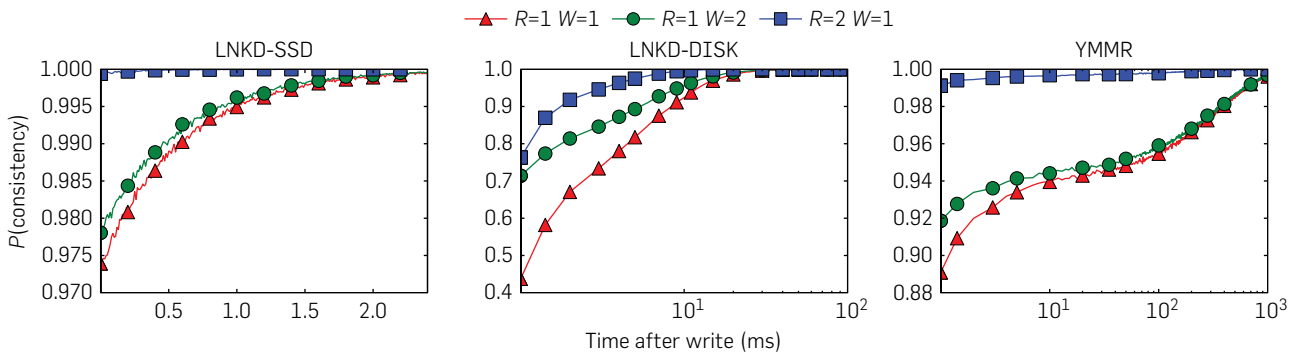
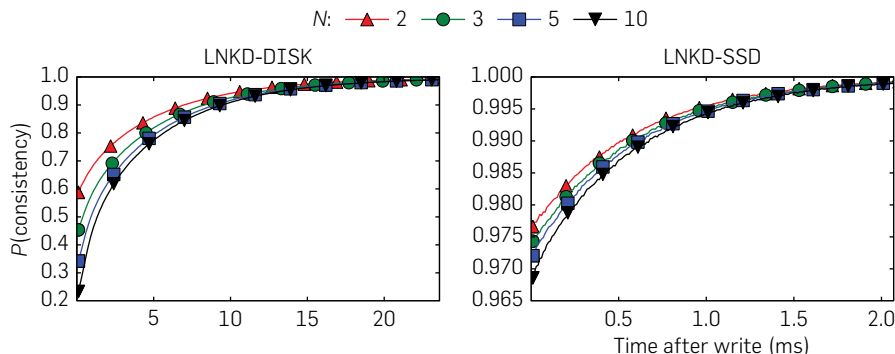


图 6 用于生产操作延迟的 (Δ, p) -正规语义, 其中 $R=W=1$, 但副本的数量 N 不同即使复制的因子相当大, 不一致性 p 也可能很快消解 (Δ 低)。



只有 21.1%。然而，在高概率 (p) 时，对于更大的副本规模，不一致的窗口 (Δ) 是关闭的。对于 LNKD-DISK, $p = 99.9\%$ 时, Δ 的范围包括从 2 个副本时的 45.3 ms 至 10 个副本时的 53.7 ms。

这些结果说明，为获取可用性或更高的性能而维护大量的副本可能会对紧接着写入操作后的一致性造成相当大的影响。不过， (Δ, p) -正规语义的概率 (p) 仍然会迅速上升 (小 Δ)。

5.6. 延迟与过期性之比较

选择 R 和 W 的值时，需要权衡操作延迟和一致性。为了度量这种权衡，我们比较了第 99.9 个百分位的操作延迟和对应的 $p = 99.9\%$ 时的 Δ ，其中仲裁配置为 $N = 3$ ，在实际中是一种典型的部署方式。

部分仲裁往往能呈现令人满意的延迟和一致性的权衡 (见表 1)。对于 YMMR 而言， $R = W = 1$ 时，会获得低延迟的读取和写入 (16.4 ms)，但 Δ 相当高 (1364 ms)。不过，设置 $R = 2$ 、 $W = 1$ 时， Δ 会降低到 202 ms，读取和写入的综合延迟比最快的严格仲裁 ($W = 1$ 、 $R = 3$) 低 81.1% (186.7 ms)。如果允许 $p = 99.9\%$ 、 $\Delta = 13.6$ ms，则 LNKD-DISK 读取和写入延迟会降低 16.5% (2.48 ms)。对于 LNKD-SSD 而言，在 10 M 次写入操作 (“七个九”) 中，取 $R = 2$ 、 $W = 1$ 时，我们没有观察到过期性。 $R = W = 1$ 会把延迟降低 59.5% (1.94 ms)，相应的 $\Delta = 1.85$ ms。总之，降低 R 和 W 的值可以极大地改进操作延迟；而且，即使是在尾部，不一致的持续时间 (Δ) 也相对较小。

本文中，我们省略了完整的结果，但我们已经针对异构的副本行为进行了实验，也针对多条目的保证 (如因果一致性和事务的原子性) 进行了实验。⁷

6. 相关讨论和未来的工作

在本节中，我们讨论了 PBS 的设计决策，描述了我们如何将 PBS 与真实世界的存储和终端用户应用进行集成，并提出了未来的研究领域。

6.1. 预测和验证

在本文中，我们开发了多种技术用于一致性的预测。给定系统的输入数据集和当前的操作环境后，它们可以提供期望的系统行为。给定轨迹后，人们可以预测在任意长度的时间或版本数量后出现的过期性，而不需要实际执行额外的查询。不仅如此，预测还能让用户轻松地执行“假设”分析，获取在任意的复制配置、请求分发 (Δ 和 K) 和硬件配置条件下的结果 (例如，从固态硬盘切换到旋转硬盘)。我们发现，预测的计算并不昂贵，它能以分散的方式基于每个副本的具体情况执行。虽然预测较为灵活，但它的好坏程度只能达到输入轨迹的好坏程度。如果使用代表性的输入数据，那么预测是准确的。然而，如果使用不具代表性的数据 (或不良的模型)，那么预测的准确度会受到影响。

相比之下，验证²¹ 会告诉用户，在确保了确定性时，他们的数据存储如何运行。如果用户使用预测器改变他们的副本设置，她可能想确保改变后的行为与预期一致。虽然此类验证并非特别适合“假设”分析，但它仍然是预测的一个重要补充。验证有效地提供了一种度量指标，这种指标源于集成 (K, Δ, p) -正规语义的密度函数，其中依据给定的读取请求的速率进行了加权 (通过最后写入后经历的时间进行测量)。不仅如此，虽然验证在算法上相当复杂¹¹，但根据我们的经验，实现起来并不是遥不可及。

我们相信，随着各个系统开始把一致性当成持续性的量化指标，上述两种技术的作用将会越来越大。整体而言，一致性预测和验证技术组成了一个功能强大的工具包。

6.2. 白盒和黑盒方法

在本文中，我们采用了白盒方法分析不一致性，并利用复制协议的专业知识提供定量方面的洞察。这要把以用户为中心的、声明式的一致性异常规范转变为后端的协议事件 (例如，在 WARS 模型中，读取和写入响应

表 1 ($\Delta, p = 99.9\%$) - 正规语义和第 99.9 个百分位处读取 (L_r) 和写入延迟 (L_w) 对应的 Δ 值，其中 R 和 W 可变， $N = 3$

	LNKD-SSD			LNKD-DISK			YMMR		
	L_r	L_w	t	L_r	L_w	t	L_r	L_w	t
$R = 1, W = 1$	0.66	0.66	1.85	0.66	10.99	45.5	5.58	10.83	1364.0
$R = 1, W = 2$	0.66	1.63	1.79	0.65	20.97	43.3	5.61	427.12	1352.0
$R = 2, W = 1$	1.63	0.65	0	1.63	10.9	13.6	32.6	10.73	202.0
$R = 2, W = 2$	1.62	1.64	0	1.64	20.96	0	33.18	428.11	0
$R = 3, W = 1$	4.14	0.65	0	4.12	10.89	0	219.27	10.79	0
$R = 1, W = 3$	0.65	4.09	0	0.65	112.65	0	5.63	1870.86	0

重要的延迟 - 过期性权衡用粗体标出。

的重排序)。另外,我们本来也可以尝试对系统内部进行反向工程,或提供一个不依赖于具体实现的预测器,但是这种黑盒分析将会变得极为复杂。我们相信,验证技术更适于黑盒技术,但必须对黑盒技术所带来的概率收益从其潜在的不准确性角度进行权衡。从我们把预测与现有数据存储进行集成的经验以及大规模采用开源数据存储的情况来看,我们相信白盒技术是可行的,即便他们需要对现有的存储进行修改。

6.3. 真实世界的存储集成

在几位开源软件开发者的帮助下,我们开发了在下列两个 NoSQL 存储系统中使用 PBS 功能的补丁: **Cassandra** 和 **Voldemort**。对于 **Cassandra**,我们采取了两种方法:一种是侵入式的、但更为准确的实现,另一种是外部的、但准确度稍低的预测模块。对于前一种方法,我们修改了 **Cassandra** 的消息层,加入了消息创建的时间戳,以便测量 **W**、**A**、**R**、**S** 的单独分布。在给定的服务器上启用跟踪后,消息层在独立的 PBS 预测模块中记录了每次操作的时间戳作为日志。时间戳储存在内存中的循环缓冲区内,用于处理各种所需的消息延迟。接下来,用户可以通过外部可访问的接口调用 PBS 预测器模块。利用该接口后,他们还能提供更高级的功能,比如动态副本配置和监测(见下)。这提供了相对准确的预测,其代价是必须调试消息层。不过,像 **Cassandra** 这样的数据存储系统已经扩展了用户可访问的监测数据(例如,每次查询的延迟跟踪)。最近,我们一直在探索在数据库之外进行预测——即后一种方法——我们实现并应用于对 **Voldemort** 的分析。我们为这两种方法都开发了开源的实现。

6.4. PBS 应用

如果没有一致性量化标准,某些功能就无法实现。而 PBS 让这些功能成为了现实。拥有 PBS 后,通过设定过期性和最低稳定性的约束,优化操作延迟,我们可以自动地配置复制参数。数据存储的操作员随后可以为应用提供服务水平协议,并用量化的方式向用户描述延迟-过期性之间的均衡关系。操作员还可以使用在线的延迟测量结果动态地配置副本。这种优化还支持人们分离用于稳定性的副本与用于实现低延迟和较高容量的副本。例如,操作员可以确定一个用于稳定性和可用性的最小复制因子,但也能自动增大 **N**,减少 **R** 和 **W** 恒定后的尾延迟。在 **SIGMOD 2013** 的演示中,我们详细阐述了这些可能的方法。该演示的特点为对活跃的 **Cassandra** 集群和模拟 web 服务进行实时预测。⁶

6.5. WARS 的局限性和扩展

WARS 模型存在若干局限性,也有一些潜在的扩展。在本文中,我们大致勾勒了一些要点,更为详细的讨论请阅读本文的扩展版本。^{5,7} **WARS** 只为单一的写

入和读取操作建模,所以对多写入的场景而言,它的估计有点保守。不仅如此,在我们当前的处理中,**WARS** 把每种分布当成 IID。对模型而言,这虽然不是根本性的,但却对我们从业界获取的延迟轨迹施加了限制。**WARS** 也没有捕捉额外的、常用的反熵流程的效果(例如,读-修复, **Merkle** 树交换)。对于过期性而言,它的估计可能有点保守。它也没有处理发生故障时的系统行为;当存储不同时该系统行为也不同。而且,它假设客户端与协调器服务器联系,而不是自己发出请求(例如, **Voldemort**)。我们相信,这些局限性并非根本性的,可以在白盒模型中加以解释。但是,它们在未来的工作中仍然会存在。最后,需要进行过期性检测的客户端可以采用异步的方式进行,启用推测性的读取和补偿处理。^{5,7}

7. 相关工作

本文研究基于若干相关领域:仲裁复制、一致性模型和量化一致性的方法。

在分布式系统和并行程序设计领域,如何管理复制的数据是一个被长期研究的问题。现在,有许多一致性模型可以解释人们在语义、性能和可用性之间的不同权衡情况。传统模型(比如可串行性和线性一致性)以及最近提出的模型(时间轴一致性⁸和并行快照隔离²³)均提供了“强”语义,但牺牲了高可用性,或者说,他们提供了一种能力,可以在所有副本上支持“始终启用”的响应行为。相比之下,面对高可用性和低延迟的需求,很多生产性的数据存储已经转向较弱的语义,以便在网络分区的情况中提供可用性。^{9,26}

本文的重点放在现有广泛部署的系统所提供的语义上。由于实践中“强”一致性和最终一致性相当流行(而且 **Dynamo** 风格的系统需要明确选择这两种模型之一),我们把重点大致放在了这种二分法上。不过,也存在多种其他方式,但他们仍属于“弱”模型。例如, **Bayou** 系统提供了一系列的“会话保证”,包括读己之所写和单调读一致性。²⁵ 与此类似,德克萨斯州大学奥斯汀分校最近的技术报告声称,因果一致性的某个变体是在可实现的单向收敛(最终一致性)系统中可以达到最强的一致性模型,¹⁸。最近这个模型已经吸引了研究人员着手系统实现。¹⁷ 正如我们上文中暗示的那样,概率的方法也可以用于本文未探讨的一致性模型。特别对于过期性而言,以前的研究,比如 **TACT**²⁷,已经详细阐述了如何提供确定性的过期性边界。这种边界确定的过期性系统在确定性方面可与 PBS 媲美。

我们使用的用于分析现代部分仲裁系统的技术吸收了理论性较强的现有文献。在第 3 节中,我们简要地概述了仲裁复制²⁰。在本文中,我们特意利用了源于概率仲裁中的灵感来¹⁹分析扩展仲裁系统及其一致性。我们认为,在写入操作传播、反熵和 **Dynamo** 的背景下再次研究概率仲裁系统——包括非多数仲裁系

统（如树仲裁）——在理论工作方面前景广阔。我们研究了过期性的概率保证，而之前对于 k -仲裁^{2,3} 的研究已经检查了确定性保证，即部分仲裁系统的返回值得属于最近写入的 k 个版本。²

最后，最近的研究着重于从理论和实验两个角度测量和验证最终一致性系统的一致性（Rahman 等人提供了简要的概述²¹）。这有助于我们验证一致性的预测并理解违反过期性的情况。

8. 结论

在本文中，我们介绍了 PBS，它可用于为最终一致的数据存储所返回的数据构建预期过期性模型。通过提供 SLA-风格的一致性预测，PBS 为很多现有系统提供了另一种保证，可用于替代全有-或-全无的一致性保证。把先前的理论扩展到概率仲裁系统上后，我们推导出了分析性的解决方案用于分析部分仲裁系统的 (K, p)-正规语义，其中读取操作的预期过期性以版本的形式呈现。我们还分析了 Dynamo 风格的仲裁复制中的 (Δ, p)-正规语义，或者读取操作在实时方面的预期过期性。为了实现上述分析，我们开发了 WARS 延迟模型来解释 Dynamo 中的消息重排序如何导致过期性。为了仔细观察实践中 (Δ, p)-正规语义的延迟影响，我们使用了来自互联网公司的真实世界轨迹来驱动蒙特卡罗分析。我们发现，最终一致的 Dynamo 风格仲裁配置往往能在几十毫秒后达到一致，主要原因是他们能适应每台服务器的延迟方差。我们得出结论，在操作无故障时，最终一致的部分仲裁复制方案不仅经常返回一致的数据，还能提供相当大的延迟收益。我们认为，随着研究人员不断地研究和部署量化一致性度量，实用的终端用户功能将得以实现，同时之前模糊和饱受争议的复制配置也会变得清晰明了。

互动的演示

关于 Dynamo 风格的 PBS 的互动演示，请访问 <http://pbs.cs.berkeley.edu/#demo>。

鸣谢

本文大大受益于前文提到的众多个人的反馈。^{5,7} 同时，它还得到了下列公司的无偿支持：谷歌、SAP、亚马逊云计算服务、Blue Goji、Cloudera、爱立信、通用电气、惠普、华为、IBM、英特尔、MarkLogic、微软、日本电气公司实验室、NetApp、NTT 多媒体通信实验室公司、Oracle、Quanta、Splunk 和 VMware。本文材料基于下列基金会的资助项目：美国国家科学基金会研究生助研奖学金（项目号 DGE 1106400）、美国国家科学基金会（资助号 IIS-0713661、CNS-0722077 和 IIS-0803690）、NSF CISE 探索（NSF CISE Expeditions）（项目号 CCF-1139158）、美国空军科学研究办公室（项目号 FA95500810352）和美国国防高级研究计划局（合同号 FA865011C7136）。



参考资料

- Abadi, D.J. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Comput.* 45, 2 (2012), 37–42.
- Aiyer, A., Alvisi, L., Bazzi, R.A. On the availability of non-strict quorum systems. In *DISC 2005*.
- Aiyer, A.S., Alvisi, L., Bazzi, R.A. Byzantine and multi-writer k -quorums. In *DISC* (2006), 443–458.
- Bailis, P., Ghodsi, A. Eventual consistency today: Limitations, extensions, and beyond. *ACM Queue* 11, 3 (Mar. 2013), 20:20–20:32.
- Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. Probabilistically bounded staleness for practical partial quorums. *PVLDB* 5, 8 (2012), 776–787.
- Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. PBS at work: Advancing data management with consistency metrics. In *SIGMOD 2013 Demo*.
- Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. Quantifying eventual consistency with PBS. *Vldb J.* (2014). (see <http://link.springer.com/article/10.1007/s00778-013-0330-1>).
- Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.A., Puz, N., Weaver, D., Yerneni, R. Pnuts: Yahoo!’s hosted data serving platform. In *Vldb 2008*.
- Davidson, S., Garcia-Molina, H., Skeen, D. Consistency in partitioned networks. *ACM Comput. Surv.* 17, 3 (1985), 314–370.
- DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Voshall, P., Vogels, W. Dynamo: Amazon’s highly available key-value store. In *SOSP 2007*, 205–220.
- Golab, W., Li, X., Shah, M.A. Analyzing consistency properties for fun and profit. In *PODC* (2011), 197–206.
- Hamilton, J. Perspectives: I love eventual consistency but... <http://perspectives.mvdirona.com/2010/02/24/ILoveEventualConsistencyBut.aspx> (24 Feb. 2010).
- Herlihy, M., Wing, J.M. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (1990), 463–492.
- Kirkell, J. Consistency or bust: Breaking a Riak cluster. <http://www.oscon.com/oscon2011/public/schedule/detail/19762>. Talk at O’Reilly OSCON 2011 (27 Jul. 2011).
- Linden, G. Make data useful. <https://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-29.ppt> (29 Nov. 2006).
- Linden, G. Marissa Mayer at Web 2.0. <http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html> (9 Nov. 2006).
- Lloyd, W., Freedman, M.J., Kaminsky, M., Andersen, D.G. Stronger semantics for low-latency geo-replicated storage. In *NSDI 2013*.
- Mahajan, P., Alvisi, L., Dahlin, M. Consistency, Availability, Convergence. Technical Report TR-11-22, Computer Science Department, University of Texas at Austin, 2011.
- Malkhi, D., Reiter, M., Wool, A., Wright, R. Probabilistic quorum systems. *Inform. Commun.* 170 (2001), 184–206.
- Merideth, M., Reiter, M. Selected results from the latest decade of quorum systems research. In *Replication*, B. Charron-Bost, F. Pedone, and A. Schiper, eds. Volume 5959 of *LNCS* (2010). Springer, 185–206.
- Rahman, M., Golab, W., AuYoung, A., Keeton, K., Wylie, J. Toward a principled framework for benchmarking consistency. In *Proceedings of the 8th Workshop on Hot Topics in System Dependability* (Hollywood, CA, 2012), USENIX.
- Schurman, E., Bruttig, J. Performance related changes and their user impact. Presented at *Velocity Web Performance and Operations Conference* (San Jose, CA, Jun. 2009).
- Sovran, Y., Power, R., Aguilera, M.K., Li, J. Transactional storage for geo-replicated systems. In *SOSP 2011*.
- Stonebraker, M. Urban myths about SQL. http://voltdb.com/_pdf/VoltDB-MikeStonebraker-SQLMythsWebinar-060310.pdf. VoltDB Webinar (Jun. 2010).
- Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M.J., Theimer, M.M., Welch, B.B. Session guarantees for weakly consistent replicated data. In *PDIS 1994*.
- Vogels, W. Eventually consistent. *CACM* 52 (2009), 40–44.
- Yu, H., Vahdat, A. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst.* 20, 3 (2002), 239–282.

Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein and Ion Stoica ({pbailis, shivaram, franklin,

hellerstein, istoica}@cs.berkeley.edu) 来自加州大学伯克利分校。

译文责任编辑：陈海波

Quantifying Eventual Consistency with PBS

By Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica

Abstract

Data replication results in a fundamental trade-off between operation latency and consistency. At the weak end of the spectrum of possible consistency models is eventual consistency, which provides no limit to the staleness of data returned. However, anecdotally, eventual consistency is often “good enough” for practitioners given its latency and availability benefits. In this work, we explain this phenomenon and demonstrate that, despite their weak guarantees, eventually consistent systems regularly return consistent data while providing lower latency than their strongly consistent counterparts. To quantify the behavior of eventually consistent stores, we introduce Probabilistically Bounded Staleness (PBS), a consistency model that provides expected bounds on data staleness with respect to both versions and wall clock time. We derive a closed-form solution for version-based staleness and model real-time staleness for a large class of quorum replicated, Dynamo-style stores. Using PBS, we measure the trade-off between latency and consistency for partial, non-overlapping quorum systems under Internet production workloads. We quantitatively demonstrate how and why eventually consistent systems frequently return consistent data within tens of milliseconds while offering large latency benefits.

1. INTRODUCTION

Modern distributed data stores need to be scalable, highly available, and fast. These systems typically replicate data across different machines and increasingly across datacenters for at least two reasons: first, to provide availability when components fail and, second, to provide improved performance by serving requests from multiple replicas. Configuring and maintaining replicated data has significant consequences for application and data store design.¹ Performance at scale is critical for a large class of applications and, in practice, increased latencies may correspond to large amounts of lost revenue.²² For example, at Amazon, 100 ms of additional latency resulted in a 1% drop in sales,¹⁵ while 500ms of additional latency in Google’s search resulted in a corresponding 20% decrease in traffic.¹⁶ However, lowering latency in distributed data stores has a cost: contacting fewer replicas for each operation can adversely impact achievable semantic guarantees.

To provide predictably low latency, modern systems often eschew protocols guaranteeing “strong” consistency of reads (e.g., the illusion of a single copy of replicated data) and instead opt for “weaker” semantics, frequently in the form of *eventual consistency*.^{1, 5, 7, 10} This eventual consistency is one of the weakest properties

provided by modern stores: it provides no guarantees on data staleness except that, in the absence of new writes, reads will “eventually” return the effect of the most recent write(s).²⁶ Under this definition, a store that returns data that is weeks old is eventually consistent, as is a store that returns arbitrary data (e.g., always return value 42) as long as, at *some point* in the future, the store returns the last written data.⁴ Due to this near-absence of useful semantics for end users, the decision to employ eventual consistency is often controversial.^{12, 23, 24} In the many production stores providing eventual consistency today,^{10, 14} users have little to no insight into the behavior of their stores or the consistency of their data, especially under varying replication configurations. However, the proliferation of eventually consistent deployments suggests that applications can often tolerate occasional staleness and that data tends to be “fresh enough” in many cases.

In this work, we bridge this gap between theoretical guarantees and current practice by quantifying the degree to which eventual consistency is both eventual and (in) consistent and explain why. Indeed, under worst-case conditions, eventual consistency results in an unbounded degree of data staleness. However, as we will show, the common case is often different. Core to our thesis is the observation that eventual consistency can be modeled as providing a probabilistic *expectation* of consistency given a particular workload and deployment environment. Accordingly, for varying degrees of certainty, eventually consistent stores can offer bounds on how far they may deviate from strongly consistent behavior. We present probabilistic models for such bounds called Probabilistically Bounded Staleness, or PBS.

To predict consistency with PBS, we need to know when and why eventually consistent systems return stale data and how to quantify the staleness of the data they return. In this paper, we present algorithms and models for two common staleness metrics in the literature: wall clock time²¹ and versions.²⁷ PBS describes both measures, providing the probability of reading a write Δ seconds after

The original version of this paper is entitled “Probabilistically Bounded Staleness for Practical Partial Quorums” and was published in *VLDB 2012*.⁵ An invited extended version of this paper is entitled “Quantifying Eventual Consistency with PBS” and will appear in the *VLDB Journal*’s “Best of VLDB 2012” issue in 2014.⁷ Portions of this work also appear in a *SIGMOD 2013* demo entitled “PBS at Work: Advancing Data Management with Consistency Metrics.”⁶

the write returns $((\Delta, p)$ -semantics, or “how eventual is eventual consistency?”), of reading one of the last K versions of a data item $((K, p)$ -semantics, or “how consistent is eventual consistency?”), and of experiencing a combination of the two $((K, \Delta, p)$ -semantics). PBS does not propose new mechanisms to enforce deterministic staleness bounds;²⁷ instead, our goal is to provide a lens for analyzing, improving, and predicting the behavior of *existing*, widely deployed systems.

In this work, we apply PBS to quorum-replicated data stores such as Dynamo¹⁰ and its several open-source descendants. Quorum systems ensure strong consistency across reads and writes to replicas by ensuring that read and write replica sets overlap. However, employing *partial* (or non-strict) quorums can lower latency by requiring fewer replicas to respond. With partial quorums, sets of replicas written to and read from need not overlap: given N replicas and read and write quorum sizes R and W , partial quorums imply $R + W \leq N$. For partial quorums, we derive closed-form solutions for PBS (K, p) -regular semantics and use Monte Carlo methods to explore the trade-off between latency and (Δ, p) -regular semantics.

Finally, we use PBS to study the staleness observed in production deployments of Dynamo-style data stores under normal operation (i.e., failure-free scenarios). We show how high variance in write latency can lead to an increased window of inconsistency. For example, in one production environment, switching from spinning disks to solid-state drives dramatically improved expected consistency (e.g., 1.85 ms versus 45.5 ms wait time for a 99.9% probability of consistent reads) due to decreased write latency mean and variance. We also make quantitative observations of the latency-consistency trade-offs offered by partial quorums. For example, in another production environment, PBS calculations show an 81.1% combined read and write latency improvement at the 99.9th percentile (230 to 43.3 ms) for a 202-ms window of inconsistency (99.9% probability consistent reads). This analysis helps demonstrate the performance benefits that lead operators to choose eventual consistency and motivates additional end-user applications ranging from consistency monitoring to consistency-based service-level agreements (SLAs) and query planning.

2. PROBABILISTICALLY BOUNDED STALENESS

By popular definitions, eventually consistent data stores do not make any guarantees as to the recency of data that they return: “if no new updates are made to the object, eventually all accesses will return the last updated value.”²⁶ This is a useful *liveness* property, guaranteeing that something good eventually happens, but it provides no *safety* properties: the data store can return any data in the interim.⁴ Many real-world eventually consistent stores do not make any guarantees beyond this definition, yet they are widely deployed and have seen increased popularity over the past decade (Section 3.2). As we will see, the protocols used by most eventually consistent stores indeed do not *enforce* additional guarantees, but they may *supply* them during operation.

To quantify semantics that are not guaranteed but are often provided, we develop a probabilistic framework for reasoning about consistency, called Probabilistically Bounded Staleness, or PBS. There are a wide range of possible consistency models that a data store can provide, from linearizability to causal consistency to eventual consistency; what is the likelihood that an end user will observe a given consistency model if it is not guaranteed? Here, we study variants of two classic kinds of (in)consistency in the form of staleness: versions and time. We develop variants of PBS metrics that provide quantitative expectations that a store will return a version that was written within the last K writes of the latest (where $K = 1$ is latest) and that a store will return the latest version as of Δ seconds ago. Ultimately, this does *not* provide a guarantee, but it is still useful for reasoning about and introspecting the behavior of a given system, similar to the usage of modern SLAs for performance (Section 6).

To begin, we first need to define a baseline for “strongly consistent” semantics. The Dynamo-style stores we study provide a choice between “strong” *regular* semantics and eventual consistency; we subsequently observe when the eventually consistent choices behave like their “strong” counterparts. According to the distributed systems literature:²

DEFINITION 1. *A read from a given data item obeys regular semantics if, in the case that the read does not overlap (in real time) with any writes to the same item, it returns the result the last completed write, or, if the read overlaps (in real time) with at least one write to the same data item, it returns either the result of the last completed write or the eventual result of one of the overlapping writes.*

Accordingly, regular semantics provide the illusion of a single copy of each replicated data item, except when there are concurrent reads and writes, during which writes’ effects may become visible and subsequently “disappear.” While this special, overlapping case is somewhat awkward to reason about (cf. definitions of linearizability¹³), this is a widely deployed data replication configuration.

Before we consider PBS applied to regular semantics, we first present a generalization of the semantics to account for multi-version staleness:²

DEFINITION 2. *A read from a given data item obeys K -regular semantics if, in the case that the read does not overlap (in real time) with a write to the same data item, it returns the result of one of the latest K completed writes, or, if the read overlaps (in real time) with a write to the same item, it returns either the result of one of the latest K completed writes or the eventual result of one of the overlapping writes.*

K -regular semantics are useful for reasoning about *how stale* a given read can be and can also be used to enforce additional consistency properties such as monotonic reads, where reads do not appear to “go back in time.”^{5,25}

We now present our first application of PBS principles. Given a system that does not guarantee K -regular

semantics, we can reason about its semantics by taking a probabilistic approach:

DEFINITION 3. A system provides (K, p) -regular semantics if each read provides K -regular semantics with probability p .

This modification is simple and straightforward: given an existing semantic guarantee, we can consider a probabilistic version of it, whereby reads may or may not obey the property. Of course, the above is simply a definition, and we must still determine how to actually provide PBS predictions, which will be the focus of the rest of the paper.

In addition to considering version-based staleness, we can also consider staleness with respect to real time. As we will see, message propagation and processing delays can influence consistency across time, so we extend regular semantics to consider time as well:

DEFINITION 4. A read obeys Δ -regular semantics if it returns either the result of the latest write as of up to Δ time units ago, the result of a write that was started but not completed as of up to Δ time units ago.

We can similarly extend this definition to a probabilistic context:

DEFINITION 5. A system provides (Δ, p) -regular semantics if each read obeys Δ -regular semantics with probability p .

Although we do not consider them in this paper, it is possible to consider both time and version staleness together, which we term (K, Δ, p) -semantics.^{5,7}

3. QUORUM SYSTEM BACKGROUND

With PBS metrics in hand, we can proceed to apply them to real systems, where they are most useful. In this study, we will consider PBS metrics in the context of quorum-replicated data stores, which represent a large class of widely deployed real-world distributed data stores. However, with some work, we believe that our methodology is also applicable to other styles of replication. Here, we provide background on quorum systems, with a focus on current practice.

3.1. Quorum foundations: Theory

Quorum systems have a long tradition as a replication strategy for distributed data.²⁰ Under quorum replication, a data store writes a data item by sending it to a set of servers responsible for the *replicas*, called a write quorum. To serve reads, the data store fetches the data from a possibly different set of replicas, called a read quorum. For reads, the data store compares the set of values returned by the replicas, and, given a total ordering of versions, can return the most recent value (or all values received, if desired). For each operation, the data store chooses (read or write) quorums from a set of sets of replicas, called a *quorum system*, with one system per data item. There are many kinds of quorum systems, but one simple configuration is to use read and write

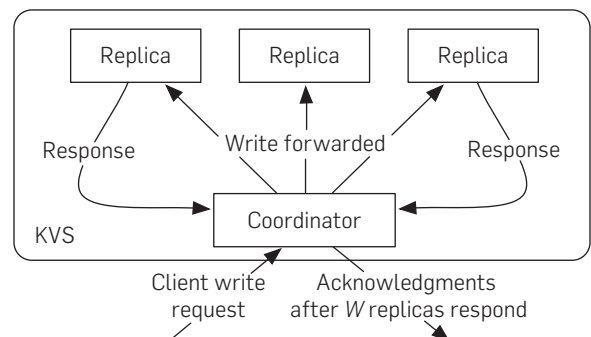
quorums of fixed sizes, which we will denote R and W , for a set of replicas of size N . A *strict quorum system* has the property that any two quorums in the quorum system overlap (have non-empty intersection), providing regular semantics. A simple example of a strict quorum system is the majority quorum system, in which each quorum is of size $\lceil \frac{N+1}{2} \rceil$. *Partial quorum systems*, which we will study, are a natural relaxation of strict quorum systems: at least two quorums in a partial quorum system do not overlap.¹⁹

3.2. Quorum foundations: Practice

In practice, many distributed data management systems use quorums as a replication mechanism. Amazon's Dynamo¹⁰ is the progenitor of a class of eventually consistent data stores that include Apache Cassandra,^a Basho Riak,^b and Project Voldemort.^c All of these systems use the same variant of quorum-style replication and we are not aware of any widely adopted data store that uses a substantially different quorum-based replication protocol.

Dynamo-style quorum systems employ one quorum system per data item, typically maintaining the mapping of items to quorum systems using a consistent-hashing scheme or a centralized membership protocol. Each server in the system cluster stores multiple items. As shown in Figure 1, clients send read and write requests to a server in the system cluster, which subsequently forwards the request to *all* replicas for the operation's item. This coordinating server considers an operation complete when it has received responses from a predetermined number of replicas (typically configured per-operation). Accordingly, without message loss, all replicas eventually receive all writes. Dynamo denotes the replication factor of an item as N , the number of replica responses required for a successful read as R , and the number of replica acknowledgments required for a successful write as W . Like other strict quorum systems, Dynamo provides regular semantics when $R + W > N$ during failure-free operation. However, unlike traditional

Figure 1. Diagram of control flow for client write to Dynamo-style quorum ($N = 3$, $W = 2$). A coordinator server handles the client write and sends it to all N replicas. The write call returns after the coordinator receives W acknowledgments.



^a <http://cassandra.apache.org/>

^b <http://www.basho.com/riak/>

^c <http://www.project-voldemort.com/>

quorum systems, Dynamo's write quorum size increases even after the operation returns, growing via *anti-entropy*.^{4,10} Coordinators send all requests to all replicas but consider only the first R (W) responses. As a matter of nomenclature (and to disambiguate against "dynamic" quorum membership protocols), we will refer to these systems as *expanding partial quorum systems*.

As we discuss in extended versions of this paper,^{5,7} system operators often report using partial quorum configurations in Dynamo-style stores, citing "maximum performance" in the "general case," particularly for "low value" data or queries that need "very low latency and high availability."

4. PBS AND PARTIAL QUORUMS

Given our PBS metrics and an understanding of quorum behavior, we can develop models for the probability of consistency under partial quorums. Here, we briefly discuss version-based staleness for traditional probabilistic quorums and develop a more complex "white box" model of time-based staleness for Dynamo-style systems.

4.1. PBS (K, p)-regular semantics

To understand static, non-expanding quorum behavior, we first revisit probabilistic quorum systems,¹⁹ which provide probabilistic guarantees of quorum intersection in partial quorum systems. As an example, consider N replicas with read and write quorums of sizes R and W chosen uniformly at random. We can calculate the probability that the read quorum does not contain the last written version. This probability is the number of quorums of size R composed of replicas that were not written to in the write quorum divided by the number of possible read quorums:

$$p_s = \frac{\binom{N-W}{R}}{\binom{N}{R}} \quad (1)$$

The probability of inconsistency is high for small values of N . However, by scaling the number of replicas and quorum sizes, one can achieve an arbitrarily high probability of consistency.¹⁹ For example, with $N = 3$, $R = W = 1$, $p_s = 0.6$, but with $N = 100$, $R = W = 30$, $p_s = 1.88 \times 10^{-6}$.² This is reminiscent of the Birthday Paradox: as the number of replicas increases, the probability of non-intersection between any two quorums decreases. Hence, the asymptotics of these systems are excellent—but only at asymptotic scales.

While probabilistic quorums allow us to determine the probability of returning the most recent value written to the database, they do not describe what happens when the most recent value is not returned. Here, we determine the probability of returning a value within a bounded number of versions ((K, p) -regular semantics). In the following formulation, we consider traditional, non-expanding write quorums (no anti-entropy).

Similar to the previous example, given independent, identically distributed (IID) choice of read and write quorums, returning one of the last k written versions is equivalent to

intersecting one of k independent write quorums. Given the probability of a single quorum non-intersection p , the probability of non-intersection with one of the last k independent quorums is p^k . Thus, the probability of non-intersection with uniform random choice in our example quorum system is Equation 1 exponentiated by k :

$$p = \left(\frac{\binom{N-W}{R}}{\binom{N}{R}} \right)^k \quad (2)$$

When $N = 3$, $R = W = 1$, this means that the probability of returning a version within 2 versions is 0.5; within 3 versions, 0.703; 5 versions, >0.868; and 10 versions, >0.98. When $N = 3$, $R = 1$, $W = 2$ (or, equivalently, $R = 2$, $W = 1$), these probabilities increase: $k = 1 \rightarrow 0.6$, $k = 2 \rightarrow 0.8$, and $k = 5 \rightarrow >0.995$.

This closed-form solution holds for quorums that do not change size over time. For expanding partial quorum systems, this solution is a lower bound on p for (K, p) -regular semantics. In the full version of this paper, we consider more advanced formulations of this analysis, including an analysis of monotonic reads consistency, quorum load, and closed-form solutions for mixed time and versions.^{5,7}

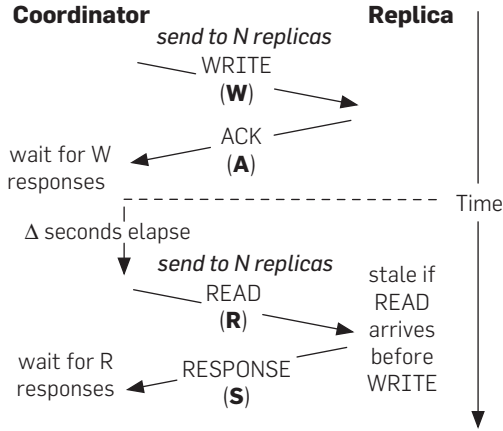
4.2. Consistency in dynamo

We have a simple, closed-form model for (K, p) -regular semantics, but time-based ((Δ, p) -regular) semantics are dependent on the quorum replication algorithm, workload, and any anti-entropy processes employed by a given system. In this section, we develop techniques for analyzing PBS (Δ, p) -regular semantics in the context of Dynamo-style data stores.

Dynamo-style quorum systems are inconsistent as a result of read and write message reordering, in turn a product of message delays. In this work, we take a white box approach to inconsistency and directly examine the protocols behind consistency. Accordingly, we develop a model of message latency in Dynamo operation that captures the effects of message delays for write requests (W), write acknowledgments (A), read requests (R), and read responses (S), and which, for convenience, we call *WARS*. In Figure 2, we illustrate *WARS* using a space-time diagram for messages between a coordinator and a single replica for a write followed by a read Δ seconds after the write completes. Accordingly, Δ here corresponds to the Δ in PBS (Δ, p) -regular semantics. In brief, reads are stale when all of the first R responses to the read request arrived at their replicas before the last (completed) write request arrived.

For a write, the coordinator sends N messages, one to each replica. The message from the coordinator to replicas containing the write is delayed by a value drawn from distribution W . The coordinator waits for W responses from the replicas before it can consider the write completed. Each response acknowledging the write is delayed by a value drawn from the distribution A .

Figure 2. The WARS model describes staleness in Dynamo by modeling message latencies between a coordinator and replicas for a write operation followed by a read operation t seconds later. In an N replica system, the depicted messages are exchanged with N replicas.



For a read, the coordinator (possibly different than the write's coordinator, and possibly representing a different client than the client that issued the write) sends N messages, one to each replica. The message from coordinator to replica containing the read request is delayed by a value drawn from distribution R . The coordinator waits for R responses from the replicas before returning the most recent value it receives. The read response from each replica is delayed by a value drawn from the distribution S .

The read coordinator will return stale data if the first R responses received reached their replicas before the replicas received the latest version (delayed by W). When $R + W > N$, this is impossible. However, under partial quorums, the frequency of this occurrence depends on the latency distributions. If we denote the write completion time (when the coordinator has received W acknowledgments) as w_t , a single replica's response is stale if $r' + w_t + \Delta < w'$ for r' drawn from R and w' drawn from W . Writes have time to propagate to additional replicas both while the coordinator waits for all required acknowledgments (A) and as replicas wait for read requests (R). Read responses are further delayed in transit (S) back to the read coordinator, inducing further possibility of reordering. Qualitatively, long-tailed write distributions (W) and relatively faster reads (R, S) increase the chance of staleness due to reordering.

WARS considers the effect of message sending, delays, and reception, but this represents a difficult analytical formulation with several non-independent order statistics. As we discuss in Section 5.1, we instead explore WARS using Monte Carlo methods, which are straightforward to understand and implement. We have found that the WARS distributions are easy to parameterize given traces of real-world system behavior (Section 5.3).

5. PBS IN ACTION

Given our PBS models for Dynamo-style stores, we now apply them to real-world systems. As discussed in Section 4.2, PBS (Δ, p) -regular behavior in a given system depends on the propagation of reads and writes across

replicas. We introduced the WARS model as a means of reasoning about inconsistency in Dynamo-style quorum systems, but quantitative metrics such as staleness observed in practice depend on each of WARS's latency distributions. In this section, we perform an analysis of Dynamo-style (Δ, p) -regular semantics for both synthetic and real-world distributions to better understand how frequently “eventually consistent” means “consistent” and, more importantly, why Dynamo-style stores are indeed frequently consistent.

PBS (K, p) -regular analysis for partial quorums is easily captured in closed form (Section 4.1). It does not depend on write latency or any environmental variables. Indeed, in practice, without expanding quorums or anti-entropy, we observe that our derived equations hold true experimentally.

In contrast, (Δ, p) -regular semantics depends on anti-entropy, which is more complicated. In this section, we focus on deriving experimental expectations for PBS (Δ, p) -regular semantics. We first validate our Monte Carlo analysis based on the WARS model and using message-level traces gathered from Cassandra clusters in Berkeley. We next explore synthetic latency distributions and, for the remainder of our analysis, explore distributions from two Internet companies: LinkedIn and Yammer.

5.1. Monte Carlo simulation

We implemented WARS analysis in a Monte Carlo-based simulation. Calculating (Δ, p) -regular semantics for a given value of Δ is straightforward (see pseudocode in Bailis et al.⁷). To simulate the message delays between coordinator and each of the N replicas, denote the i th sample drawn from distribution D as $D[i]$ and draw N samples from W, A, R , and S . Compute the time that the write request completes (w_t , or the time that the coordinator gets its W th acknowledgment; the W th smallest value of $\{W[i] + A[i], i \in [0, N)\}$). Next, determine whether any of the first R replicas contained an up-to-date response: check whether any the first R samples of R , ordered by $R[i] + S[i]$ obey $w_t + R[i] + \Delta \leq W[i]$. Repeating this process multiple times provides an approximation of the behavior specified by the trace. Extending this formulation to analyze (K, Δ, p) -regular semantics given a distribution of write arrival times will require accounting for multiple writes across time. As described in extended versions of this paper,^{5, 7} we validated this analysis on traces that we collected from a real-world Cassandra cluster. We observed an average RMSE of 0.28% for (Δ, p) -regular semantics prediction and an average N-RMSE of 0.48% for latency predictions.

5.2. Write latency distribution effects

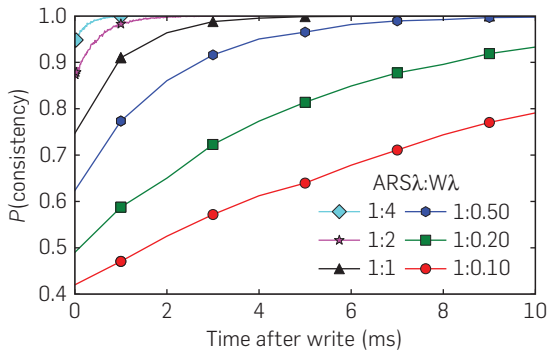
The WARS model of Dynamo-style systems dictates that high variance in latency increases staleness. Before studying real-world workloads (Section 5.3), we quantified this behavior in isolation via synthetic distributions: we swept a range of exponentially distributed write distributions (changing parameter λ , which dictates the mean and tail of the distribution) while fixing $A = R = S$.

Our results, shown in Figure 3, demonstrate this relationship. When the variance and mean of W are 0.0625 ms

and 0.25 ms ($\lambda = 4$, one-fourth the mean of $A = R = S = 1$ ms), we observe a 94% chance of consistency immediately after the write and 99.9% chance after 1 ms. However, when the variance and mean of W are 100 ms and 10 ms ($\lambda = 0.1$, ten times the mean of $A = R = S = 1$ ms), we observe a 41% chance of consistency immediately after write and a 99.9% chance of consistency only after 65 ms. As the variance and mean increase, so does the probability of inconsistency. Under distributions with fixed means and variable variances (uniform, normal), we observe that the mean of W is less important than its variance if W is strictly greater than $A = R = S$.

Decreasing the mean and variance of W improves the probability of consistent reads. This means that, as we will see, techniques that lower write latency variance result in more consistent reads. Instead of increasing read and write quorum sizes, operators could choose to lower (relative) W latencies through hardware configuration or by delaying reads, although this latter option is detrimental to performance for read-dominated workloads and may introduce undesirable queuing effects. Nonetheless, this PBS analysis illustrates the fact that stale reads can be avoided in a variety of ways beyond simple adjustment of quorum sizes.

Figure 3. (Δ, p)-regular semantics with exponential latency distributions for W and $A = R = S$. Mean latency is $1/\lambda$. $N = 3$, $R = W = 1$.



5.3. Production latency distributions

To study real-world behavior, we obtained production latency statistics from two Internet-scale companies. While message-level *WARS* timing traces would deliver more accurate predictions, we opted for a pragmatic compromise: as described in extended versions of this work, we fit *WARS* distributions to each of the provided statistics (Figure 4).^{5,7}

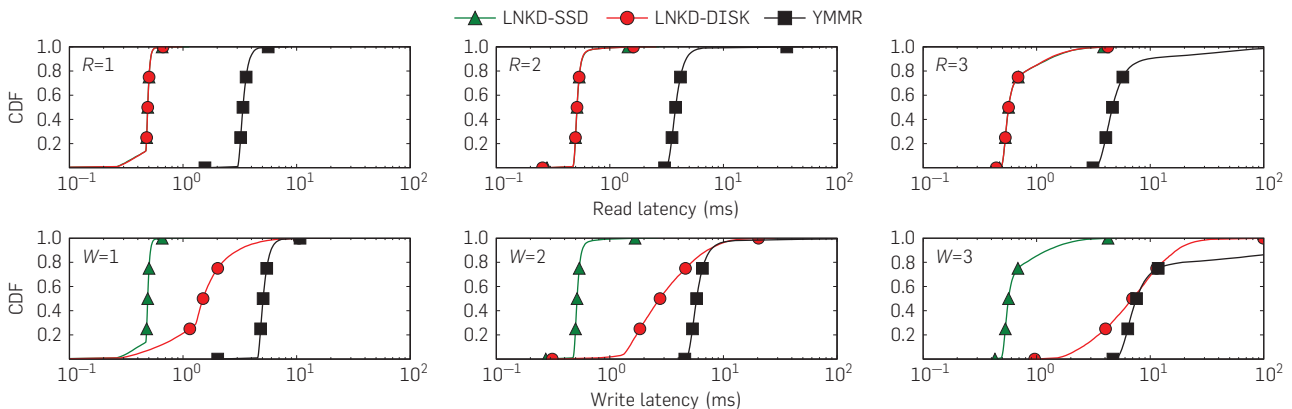
LinkedIn^d is an online professional social network with over 225 million members as of July 2013. To provide highly available, low latency data storage, engineers at LinkedIn built Voldemort. Alex Feinberg, a lead engineer on Voldemort, graciously provided us with latency distributions for a single server under peak traffic for a user-facing service at LinkedIn, representing 60% read and 40% read-modify-write traffic.^{5,7} Feinberg reports that, using spinning disks, Voldemort is “largely IO bound and latency is largely determined by the kind of disks we’re using, [the] data to memory ratio and request distribution.” With solid-state drives (SSDs), Voldemort is often “CPU and/or network bound” and “maximum latency is generally determined by [garbage collection] activity (rare, but happens occasionally) and is within hundreds of milliseconds.” We denote the LinkedIn spinning disk distribution as LNKD-DISK and SSD trace as LNKD-SSD.

Yammer^e provides private social networking to over 200,000 companies as of July 2013 and uses Basho’s Riak for some client data. Coda Hale, an infrastructure architect, provided performance statistic for their production Riak deployment.^{5,7} Hale mentioned that “reads and writes have radically different expected latencies, especially for Riak.” Riak delays writes “until the fsync returns, so while reads are often <1 ms, writes rarely are.” Also, although we do not model this explicitly, Hale also noted that the size of values is important, claiming “a big performance improvement by adding LZF compression to values.” We denote the Yammer latency distribution as YMMR.

^d <http://www.linkedin.com/>

^e <http://www.yammer.com/>

Figure 4. Read and write operation latency for production fits for $N = 3$ and varying R and W . For reads, LNKD-SSD is equivalent to LNKD-DISK. Depicted points mark significant percentiles for comparison. Higher values of R and W result in increased latency.



5.4. Staleness in production

The production latency distributions confirm that staleness is frequently limited in eventually consistent stores. We measured the (Δ, p) -regular semantics for each distribution (Figure 5). LNKD-SSD and LNKD-DISK demonstrate the importance of write latency in practice. Immediately after write completion, LNKD-SSD had a 97.4% probability of consistent reads, reaching over a 99.999% probability of consistent reads after 5 ms. LNKD-SSD's reads briefly raced with its writes immediately after write completion. However, within a few milliseconds after the write, the chance of a read arriving before the last write was nearly eliminated. The distribution's read and write operation latencies were small (median 0.489 ms), and writes completed quickly across all replicas due to the distribution's short tail (99.9th percentile 0.657 ms). In contrast, under LNKD-DISK, writes take much longer (median 1.50 ms) and have a longer tail (99.9th percentile 10.47 ms). LNKD-DISK's (Δ, p) -regular semantics reflects this difference: immediately after write completion, LNKD-DISK had only a 43.9% probability of consistent reads and, 10 ms later, only a 92.5% probability. This suggests that SSDs may greatly improve consistency due to reduced write variance. Similarly, one should expect that consistency would be improved by using explicit memory management rather than unscheduled garbage collection.

We experienced similar behavior with the other distributions. Immediately after write completion ($\Delta = 0$), YMMR had a $p = 89.3\%$ probability of consistency. However, the YMMR distribution only reached a $p = 99.9\%$ probability of consistency at $\Delta = 1364$ ms due to high variance and long-tail behavior in its write distribution. This hints that, given multiple replicas for a data item, the durability benefits of synchronously flushing writes to disk may have adverse effects on consistency. An alternative approach that could improve consistency and avoid this high variance would rely on multi-replica (in-memory or buffer cache) replication for durability and only flush writes asynchronously.

5.5. Quorum sizing

In addition to $N = 3$ —the most common quorum size we encountered in practice—we consider how varying the number of replicas (N) affects (Δ, p) -regular semantics while maintaining $R = W = 1$. The results, depicted in Figure 6, show that the probability of consistency immediately after write completion decreases as N increases. With two replicas, LNKD-DISK has a 57.5% probability of consistent reads immediately after write completion but only a 21.1% probability with 10 replicas. However, at high probabilities (p), the window of inconsistency (Δ) for increased replica sizes is close. For LNKD-DISK, Δ at $p = 99.9\%$ ranges from 45.3 ms for 2 replicas to 53.7 ms for 10 replicas.

Figure 5. (Δ, p) -regular semantics for production operation latencies.

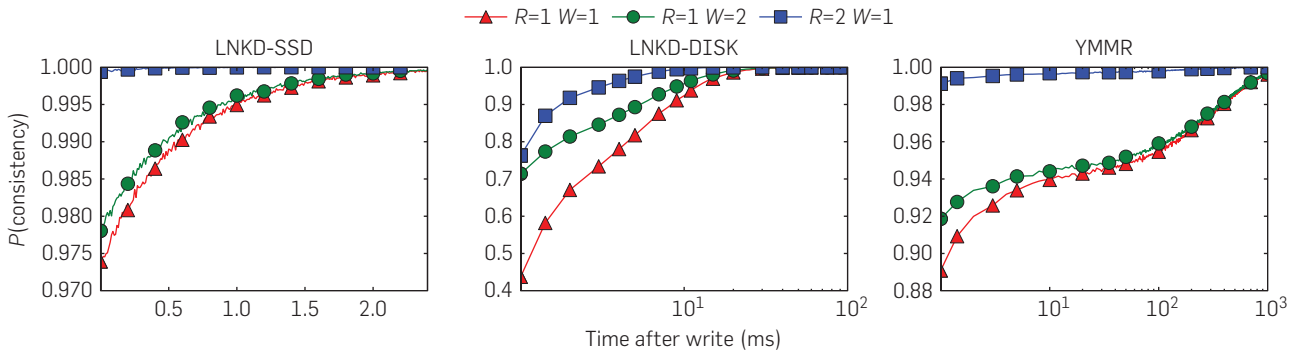
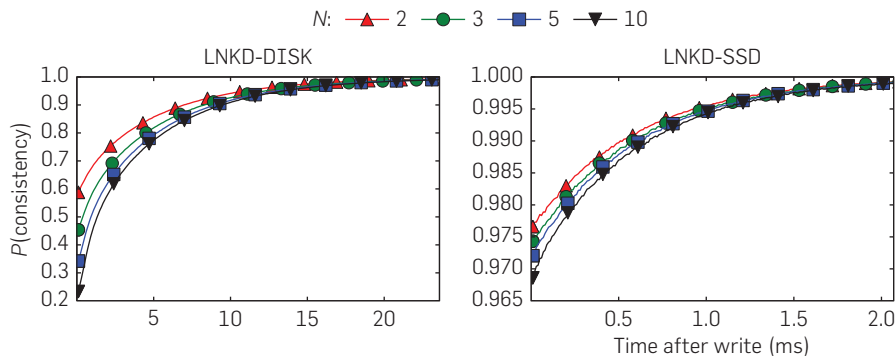


Figure 6. (Δ, p) -regular semantics for production operating latencies varying the number of replicas N when $R = W = 1$. Inconsistencies p are likely to resolve quickly (low Δ) even for large replication factors.



These results imply that maintaining a large number of replicas for availability or better performance results in a potentially large impact on consistency immediately after writing. However, the (Δ, p) -regular semantics probability (p) will still rise quickly (small Δ).

5.6. Latency vs. staleness

Choosing values for R and W is a trade-off between operation latency and consistency. To measure this trade-off, we compared 99.9th percentile operation latencies with the corresponding Δ at $p = 99.9\%$ for quorum configurations where $N = 3$, typical of deployments in the field.

Partial quorums often exhibit favorable latency-consistency trade-offs (Table 1). For YMMR, $R = W = 1$ results in low latency reads and writes (16.4ms) but high Δ (1364ms). However, setting $R = 2$ and $W = 1$ reduces Δ to 202ms and the combined read and write latencies are 81.1% (186.7ms) lower than the fastest strict quorum ($W = 1$, $R = 3$). Allowing $p = 99.9\%$, $\Delta = 13.6$ ms reduces LNKD-DISK read and write latencies by 16.5% (2.48 ms). For LNKD-SSD, across 10 M writes (“seven nines”), we did not observe staleness with $R = 2$, $W = 1$. $R = W = 1$ reduced latency by 59.5% (1.94 ms) with a corresponding $\Delta = 1.85$ ms. In summary, lowering values of R and W can greatly improve operation latency but, even in the tail, the duration of inconsistency (Δ) is relatively small.

We omit full results here, but we have also experimented with heterogeneous replica behavior and with multi-item guarantees such as causal consistency and transactional atomicity.⁷

6. DISCUSSION AND FUTURE WORK

In this section, we discuss PBS design decisions, describe our experiences integrating PBS with real-world stores and end-user applications, and suggest areas for future work.

6.1. Prediction and verification

In this work, we have developed techniques for consistency prediction, which provide an expectation of system behavior given a set of input data about the system and the current operating environment. Given a trace, one can predict staleness after an arbitrary amount of time or number of versions without having to actually run any additional queries.

Additionally, predictions allow users to easily perform “what-if” analysis across arbitrary replication configurations, request distributions (Δ and K), and hardware configurations (e.g., switching from SSDs to disks). We have found that prediction is computationally inexpensive and can be performed on a decentralized, per-replica basis. While prediction is flexible, it is only as good as the input traces. With representative input data, predictions will be accurate. However, with unrepresentative data (or bad models), prediction accuracy will suffer.

In contrast, verification²¹ informs users how their data stores are performing with guaranteed certainty. If a user makes a change to their replication settings using a predictor, she may want to ensure that the change behaves as expected. While this verification is not well-suited to “what-if” analysis, it is an important complement to prediction. Verification effectively provides a metric that results from integrating the (K, Δ, p) -regular semantics density function weighted by the given read request rate (measured with respect to time since the last write). Additionally, verification is algorithmically complex,¹¹ but in our experience is not terribly difficult to implement.

We believe that both techniques will be increasingly useful as systems begin to treat consistency as a continuous, quantitative metric. Taken together, consistency prediction and verification techniques form a powerful toolkit.

6.2. White and black box approaches

In this work, we use a white box approach to consistency and exploit expert knowledge of replication protocols to provide quantitative insight. This requires translating from user-centric, declarative specifications of consistency anomalies into back-end protocol events (e.g., in the WARS model, the reordering between read and write responses). We could have alternatively attempted to reverse engineer system internals or provide an implementation-independent predictor, but this black box analysis would be substantially more complex. We believe that verification techniques are more amenable to black box techniques and that the portability benefits of black box techniques must be weighed against their potential inaccuracy. Given our experiences integrating prediction into existing data

Table 1. Δ for $(\Delta, p = 99.9\%)$ -regular semantics and 99.9th percentile read (L_r) and write latencies (L_w), varying R and W with $N = 3$.

	LNKD-SSD			LNKD-DISK			YMMR		
	L_r	L_w	t	L_r	L_w	t	L_r	L_w	t
$R = 1, W = 1$	0.66	0.66	1.85	0.66	10.99	45.5	5.58	10.83	1364.0
$R = 1, W = 2$	0.66	1.63	1.79	0.65	20.97	43.3	5.61	427.12	1352.0
$R = 2, W = 1$	1.63	0.65	0	1.63	10.9	13.6	32.6	10.73	202.0
$R = 2, W = 2$	1.62	1.64	0	1.64	20.96	0	33.18	428.11	0
$R = 3, W = 1$	4.14	0.65	0	4.12	10.89	0	219.27	10.79	0
$R = 1, W = 3$	0.65	4.09	0	0.65	112.65	0	5.63	1870.86	0

Significant latency-staleness trade-offs are in bold.

stores and the large-scale adoption of open-source data stores, we believe that white box techniques are feasible, even if they require modifications to existing stores.

6.3. Real-world store integration

With the help of several open-source developers, we have developed patches for PBS functionality within two NoSQL stores: Cassandra and Voldemort. For Cassandra, we have taken two approaches: an invasive but more accurate implementation and an external but less accurate prediction module. For the former approach, we modified the Cassandra messaging layer to add a message creation timestamp in order to measure each of the W , A , R , S distributions. When tracing is enabled on a given server, the messaging layer logs per-operation timestamps in a separate PBS prediction module. The timestamps are stored in an in-memory circular buffer for each of the required message latencies. Subsequently, users can call the PBS predictor module via an externally accessible interface, which they can use to provide advanced functionality like dynamic replication configuration and monitoring (see below). This provides relatively accurate predictions at the expense of having to instrument the messaging layer. However, as data stores like Cassandra have expanded their user-accessible monitoring data (e.g., per-query latency tracing), we have more recently been exploring predictions outside of the database—the latter approach—which we have implemented for Voldemort. We have open-sourced implementations of both approaches.

6.4. PBS applications

PBS enables functionality not possible without quantitative consistency metrics. With PBS, we can automatically configure replication parameters by optimizing operation latency given constraints on staleness and minimum durability. Data store operators can subsequently provide service level agreements to applications and quantitatively describe latency-staleness trade-offs to users. Operators can dynamically configure replication using online latency measurements. This optimization also allows disentanglement of replication for durability from replication for reasons of low latency and higher capacity. For example, operators can specify a minimum replication factor for durability and availability but can also automatically increase N , decreasing tail latency for fixed R and W . We expanded upon these possibilities in a *SIGMOD 2013* demo that featured real-time predictions for a live Cassandra cluster and mock web service.⁶

6.5. WARS limitations and extensions

There are several limitations and potential extensions of the *WARS* model, which we sketch here and discuss in greater detail in extended versions of this work.^{5,7} *WARS* only models a single write and read and is therefore a conservative estimate for multi-write scenarios. Moreover, in our current treatment, *WARS* treats each distribution as IID. This is not fundamental to the model but is a limitation of our latency traces from industry. *WARS* also does

not capture effects of additional, commonly employed anti-entropy processes (e.g., read-repair, Merkle-tree exchange) and may be a conservative estimate of staleness. It does not address system behavior under failures, which varies from store to store, and it assumes that clients contact a coordinator server instead of issuing requests themselves (e.g., Voldemort). We believe that these limitations are not fundamental and can be accounted for in a white box model but nonetheless remain future work. Finally, clients requiring staleness detection may do so asynchronously, enabling speculative reads and compensatory actions.^{5,7}

7. RELATED WORK

This research builds upon several related areas: quorum replication, consistency models, and approaches to quantifying consistency.

Managing replicated data is a long-studied problem in distributed systems and concurrent programming. There is a plethora of consistency models offering different trade-offs between semantics, performance, and availability. Traditional models such as serializability and linearizability as well as more recently proposed models such as timeline consistency⁸ and parallel snapshot isolation²³ all provide “strong” semantics at the cost of high availability, or the ability to provide “always-on” response behavior at all replicas. In contrast, faced with a requirement for high availability and low latency, many production data stores have turned to weaker semantics to provide availability in the face of network partitions.^{9,26}

Our focus in this paper is on the semantics provided by existing, widely deployed systems. Due to the prevalence of “strong” consistency and eventual consistency models in practice (and the explicit choice between these two models in Dynamo-style systems), we largely focus on this dichotomy. However, there are a range of alternative but still “weak” models. As an example, the Bayou system provided a range of “session guarantees,” including read-your-writes and monotonic reads consistency.²⁵ Similarly, a recent Technical Report from UT Austin claims that a variant of causal consistency is the strongest consistency model achievable in an available, one-way convergent (eventually consistent) system,¹⁸ a model that has recently attracted systems implementations.¹⁷ As we have hinted, probabilistic approaches are applicable to the consistency models beyond those we have considered here. Specific to staleness, prior research such as TACT²⁷ has examined how to provide deterministic staleness bounds. These deterministically bounded staleness systems represent the deterministic dual of PBS.

Our techniques for analyzing modern partial quorum systems draw on existing, largely theoretical literature. We briefly surveyed quorum replication²⁰ in Section 3. In this work, we specifically draw inspiration from probabilistic quorums¹⁹ in analyzing expanding quorum systems and their consistency. We believe that revisiting probabilistic quorum systems—including non-majority quorum systems such as tree quorums—in the context of write propagation, anti-entropy, and Dynamo is a promising area for

theoretical work. While we study probabilistic guarantees for staleness, prior work on k -quorums^{2, 3} have looked at *deterministic* guarantees that a partial quorum system will return values that are within k versions of the most recent write.²

Finally, recent research has focused on measuring and verifying the consistency of eventually consistent systems both theoretically and experimentally (Rahman et al. provide a brief survey²¹). This is useful for validating consistency predictions and understanding staleness violations.

8. CONCLUSION

In this paper, we introduced PBS, which models the expected staleness of data returned by eventually consistent data stores. PBS offers an alternative to the all-or-nothing consistency guarantees of many of today's systems by offering SLA-style consistency predictions. By extending prior theory on probabilistic quorum systems, we derived an analytical solution for the (K, p) -regular semantics of a partial quorum system, representing the expected staleness of a read operation in terms of versions. We also analyzed (Δ, p) -regular semantics, or expected staleness of a read in terms of real time, under Dynamo-style quorum replication. To do so, we developed the *WARS* latency model to explain how message reordering leads to staleness under Dynamo. To examine the effect of latency on (Δ, p) -regular semantics in practice, we used real-world traces from Internet companies to drive a Monte Carlo analysis. We find that eventually consistent Dynamo-style quorum configurations are often consistent after tens of milliseconds due in large part to their resilience to per-server latency variance. We conclude that eventually consistent partial quorum replication schemes frequently deliver consistent data during failure-free operation while offering significant latency benefits. We believe that continued study and deployment of quantitative consistency metrics will both enable useful end-user functionality and shed light on previously opaque and frequently controversial replication configurations.

Interactive demonstration

An interactive demonstration of Dynamo-style PBS is available at <http://pbs.cs.berkeley.edu/#demo>.

Acknowledgments

This work was greatly improved by feedback from many previously noted individuals.^{5, 7} It was supported by gifts from Google, SAP, Amazon Web Services, Blue Goji, Cloudera, Ericsson, General Electric, Hewlett Packard, Huawei, IBM, Intel, MarkLogic, Microsoft, NEC Labs, NetApp, NTT Multimedia Communications Laboratories, Oracle, Quanta, Splunk, and VMware. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant DGE 1106400, National Science Foundation Grants IIS-0713661, CNS-0722077, and IIS-0803690, NSF CISE Expeditions award CCF-1139158, the Air Force Office of Scientific Research Grant FA95500810352, and by DARPA contract FA865011C7136.

References

1. Abadi, D.J. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Comput.* 45, 2 (2012), 37–42.
2. Aiyer, A., Alvisi, L., Bazzi, R.A. On the availability of non-strict quorum systems. In *DISC 2005*.
3. Aiyer, A.S., Alvisi, L., Bazzi, R.A. Byzantine and multi-writer k -quorums. In *DISC* (2006), 443–458.
4. Bailis, P., Ghodsi, A. Eventual consistency today: Limitations, extensions, and beyond. *ACM Queue* 11, 3 (Mar. 2013), 20:20–20:32.
5. Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. Probabilistically bounded staleness for practical partial quorums. *PVLDB* 5, 8 (2012), 776–787.
6. Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. PBS at work: Advancing data management with consistency metrics. In *SIGMOD 2013 Demo*.
7. Bailis, P., Venkataraman, S., Franklin, M.J., Hellerstein, J.M., Stoica, I. Quantifying eventual consistency with PBS. *Vldb J.* (2014). (see <http://link.springer.com/article/10.1007/s00778-013-0330-1>).
8. Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.A., Puz, N., Weaver, D., Yerneni, R. Phnuts: Yahoo!'s hosted data serving platform. In *Vldb 2008*.
9. Davidson, S., Garcia-Molina, H., Skeen, D. Consistency in partitioned networks. *ACM Comput. Surv.* 17, 3 (1985), 314–370.
10. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W. Dynamo: Amazon's highly available key-value store. In *SOSP 2007*, 205–220.
11. Golab, W., Li, X., Shah, M.A. Analyzing consistency properties for fun and profit. In *PODC* (2011), 197–206.
12. Hamilton, J. Perspectives: I love eventual consistency but... <http://perspectives.mdirona.com/2010/02/24/ILoveEventualConsistencyBut.aspx> (24 Feb. 2010).
13. Herlihy, M., Wing, J.M. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (1990), 463–492.
14. Kirkell, J. Consistency or bust: Breaking a Riak cluster. <http://www.oscon.com/oscon2011/public/schedule/detail/19762>. Talk at O'Reilly OSCON 2011 (27 Jul. 2011).
15. Linden, G. Make data useful. <https://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-29.ppt> (29 Nov. 2006).
16. Linden, G. Marissa Mayer at Web 2.0. <http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html> (9 Nov. 2006).
17. Lloyd, W., Freedman, M.J., Kaminsky, M., Andersen, D.G. Stronger semantics for low-latency geo-replicated storage. In *NSDI 2013*.
18. Mahajan, P., Alvisi, L., Dahlin, M. *Consistency, Availability, Convergence*. Technical Report TR-11-22, Computer Science Department, University of Texas at Austin, 2011.
19. Malkhi, D., Reiter, M., Wool, A., Wright, R. Probabilistic quorum systems. *Inform. Commun.* 170 (2001), 184–206.
20. Merideth, M., Reiter, M. Selected results from the latest decade of quorum systems research. In *Replication*, B. Charron-Bost, F. Pedone, and A. Schiper, eds. Volume 5959 of *LNCS* (2010). Springer, 185–206.
21. Rahman, M., Golab, W., AuYoung, A., Keeton, K., Wylie, J. Toward a principled framework for benchmarking consistency. In *Proceedings of the 8th Workshop on Hot Topics in System Dependability* (Hollywood, CA, 2012), USENIX.
22. Schurman, E., Brutlag, J. Performance related changes and their user impact. Presented at *Velocity Web Performance and Operations Conference* (San Jose, CA, Jun. 2009).
23. Sovran, Y., Power, R., Aguilera, M.K., Li, J. Transactional storage for geo-replicated systems. In *SOSP 2011*.
24. Stonebraker, M. Urban myths about SQL. http://voltdb.com/_pdf/Voltdb-MikeStonebraker-SQLMythsWebinar-060310.pdf. Voltdb Webinar (Jun. 2010).
25. Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M.J., Theimer, M.M., Welch, B.B. Session guarantees for weakly consistent replicated data. In *PDIS 1994*.
26. Vogels, W. Eventually consistent. *CACM* 52 (2009), 40–44.
27. Yu, H., Vahdat, A. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst.* 20, 3 (2002), 239–282.

Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica ({pbailis, shivaram, franklin, hellerstein, istoica}@cs.berkeley.edu), University of California, Berkeley.