

为读取共享数据的客户端提供更多类型的数据一致性保证措施是可行的，可能也是必须的。

作者：DOUG TERRY

以棒球赛为例 解释复制数据 的一致性问題

云中的复制存储系统为读取数据的应用程序提供了各种不同的一致性保证措施。云存储供应商总是以冗余方式将数据存储在台机器上，从而确保在发生不可避免的故障时，数据仍然可用。跨数据中心进行数据复制的情况并不少见，使得数据在某个数据中心完全瘫痪的情况下有可能幸免于难。不过，这些数据副本并不是总能保持完全同步。因此，客户端从不同的服务器读取同一数据对象时，有可能接收到不同的数据版本。

某些系统，如微软的 Windows Azure，对于应用程序只提供强一致性存储服务。⁵ 这些服务确保 Windows Azure 存储的客户端看到的总是数据对象的最新值。尽管在数据中心内部提供强一致性存储服务是一种符合用户期待的合理选择，但当系统开始为位于多个大洲的多个数据中心提供跨越地理范围的数据复制服务时，却带来了问题。

许多云存储系统，如亚马逊的“简单存储服务”（S3），在设计时采用了弱一致性。其设计者认为强一致性对于大型系统而言代价过大。他们选择松弛一致性，以获得更好的性能和可用性。客户端在此类系统中执行读操作时，可能会得到过期的数据。读操作所返回的数据，是过去某个时间点上该对象的值，但不一定是最新的值。例如，当读操作指向了一个副本，而该副本尚未像别的副本一样收到所有写操作时，就会发生这种情况。此类系统被称作支持最终一致性。¹²

近来设计的系统都认识到支持不同类型应用程序的必要性，提供了访问云存储的多种操作方式。例如，亚马逊的 DynamoDB 同时提供了最终一致性读取方式和强一致性读取方式，后者需要较长的读取等待时间，并使读取吞吐量下降一半。¹ 亚马逊 SimpleDB 系统为读取数据的客户端提供了同样的选择。与此类似，Google App Engine Datastore（谷歌应用引擎数据存储）系统新增了最终一致性读取方式，对其默认的强一致性读取方式进行补充。⁸ 雅虎公司许多 Web 服务底层所使用的 PNUTS，提供了三种类型的读操作：任意读（read-any）、临界读（read-critical）

» 重要见解

- 虽然复制云服务通常提供强一致性或最终一致性，但位于两者之间的一致性保证可能会更好地满足应用程序的需求。
- 一致性保证可以使用一种与实现方案无关的方式来定义，而且可以在每次进行读操作时加以选择。
- 对于松弛一致性的处理，不一定会给应用程序开发人员带来额外的负担。



和最新读 (*read-latest*)。⁷ 基于配额的新型存储系统允许客户端通过选择不同的读写配额来选择强一致性或最终一致性。⁴

研究界在过去 30 年中为分布式复制系统提出了若干数据一致性模型。¹⁰ 这些模型提供介于强一致性和最终一致性两者之间的一致性保证。例如，某个系统可能会

保证客户端看到的数据不会是 5 分钟之前的过时数据，或保证客户端始终看到自己写入的结果。实际上，一些一致性模型甚至比最终一致性模型还要弱。但鉴于这些模型用处不大，本文略去对它们的讨论。

之所以要探索不同的一致性模型，是因为一致性、性能和

可用性 (简称 CAP) 不能同时达到最优。^{9,10,12,13} 提供更强的一致性，通常会造成较低的读写性能和较差的数据可用性。CAP 定理已经证明，对于必须容忍网络不连通的系统，设计人员必须在一致性和可用性之间进行抉择。⁵ 在实践中，访问延迟也是同样重要的一个考虑因素。¹ 在各种复杂的利弊

表 1. 六种一致性保证措施。

强一致性	看到之前的所有写入。
最终一致性	看到之前写入的子集。
一致性前缀	看到初始写入序列。
限定过期法	看到所有“旧”的写入。
单调读	看到写入的增量子集。
读自身写	看到读取者过去的所有写操作。

权衡下，人们提出的每一种一致性模型都有某种意义。

不同的一致性有实际用处吗？应用程序开发人员能应对最终一致性吗？现有的云存储服务已经提供了一致性读方式和最终一致性读方式，系统是否应提供更多的一致性选项呢？

通过研究棒球赛这一应用示例（显然是虚构示例），本文试图回答这些问题，至少是部分回答这些问题。需要了解棒球赛得分的人包括记分员、裁判员、电台记者、体育新闻记者和统计员，下面我将详细探讨不同人员的各种需求。假设比赛得分存储在一个基于云的复制存储服务中，我要说明最终一致性对于大多数参与者来讲是不够的，而强一致性也是不必要的。大多数参与者会受益于一些中间的一致性。

下一节将定义针对读操作的六种可能的一致性保证。然后，我将给出一种模拟棒球赛的算法，其中表明写入和读取数据是在何处发生的。当采用不同的一致性保证措施读取比分时，我会将所有可能返回的结果——列举出来。我还将研究想知道棒球比分数据的这些人所担当的不同角色，以及每种角色所期望的读一致性，并从这个简单的例子中得出我的结论。

读一致性保证

尽管在过去 30 年里，复制系统提供了多种数据一致性措施，计算机科学研究界也已探索出各种各样的一致性模型，但是这些模型

很多都限定在特定的实现方案中。要理解系统在什么情况下能够提供什么样的一致性，经常需要了解系统是如何运作的。不幸的是，这把负担加到那些为此类存储系统开发应用程序的人身上。

笔者此处提倡的六种一致性保证可以用一种简单的、与实现无关的方式进行描述。这不仅能让应用程序开发人员获益，还能给底层存储系统的设计、运行和演化带来灵活性。

这些一致性保证是基于客户端要对数据存储区进行读写操作这样一个简单模型。多个客户端可能同时访问共享的信息，如社会关系图、新闻推送、照片、购物车或财务记录。数据在一组服务器间复制，而复制协议的细节对客户端是不可见的。写操作是指对一个或多个数据对象进行更新的任何操作。所有服务器最终都会收到这些写操作，并以相同的顺序加以执行。该顺序与客户端提交写操作的顺序是一致的。实际上，可以通过在主服务器上执行所有这些写操作来保证这个顺序，或者通过在每个服务器上运行一种一致性协议来就全局顺序达成一致。读操作返回先前写入一个或多个数据对象的值，但不一定是最新的值。每个读操作可以请求某种一致性保证，所选择的一致性决定了允许返回值集合。每种一致性保证由哪些先前写入的结果对读操作可见定义。表 1 总结了六种一致性保证

强一致性特别容易理解。这种一致性确保读操作返回的值就是

最后一次写入给定对象的值。如果写操作可以修改或扩充部分数据对象，例如将数据追加到日志的末尾，那么，读操作会返回将所有写操作应用到该对象后的结果。换句话说，读操作看到的是所有先前写操作的结果。

最终一致性是最弱的一种一致性保证。这意味着它的可能返回值集合最为庞大。对于整个对象的写入，最终一致性读操作可能返回曾经写入该数据对象的任何值。广而言之，此读操作可能会从一个副本返回结果，该副本可能收到了对被读取数据对象写操作的任意子集。术语“最终”一致性源于这样一个事实：每个副本最终会收到每一个写操作。如果客户端停止了写操作，那么读操作最终将返回对象的最新值。

通过请求一致性前缀，可以保证数据读取者能够看到从第一次写入数据存储区开始的一个有序的写入结果序列。例如，读操作可能从一个副本得到应答，而该副本从主副本那里依次接收写操作，但尚未收到最近的一些写操作。换句话说，数据读取者看到的是数据存储区的某个版本，而该版本是在过去某个时间存在于主副本上的。这类似于许多数据库管理系统所提供的“快照隔离”一致性。对于读取单个数据对象的操作，如果系统中写操作完全覆盖该对象以前的值，那么即便是最终一致性的读操作也能看到一致性前缀。当读取多个数据对象，或写操作增量更新一个对象时，就能体现出请求一致性前缀的好处。

限定过期法确保读取的结果不会太陈旧。典型情况下，用时间段 T 定义过期，比如五分钟。存储系统确保读操作返回不超过 T 分钟以前写入的值或是更晚写入的值。还有一些系统采用另外的方式来定义数据过期：缺失的写入次数，甚至是数据值的不准确

程度。我觉得时间限定过期法对应用程序开发人员来说是最自然的概念。

单调读 (monotonic read) 是一种属性，适用于由某一存储系统客户端执行的读操作序列。因此称之为“会话期保证”。¹¹ 对于单调读，客户端可以读取任意陈旧数据，这一点与最终一致性相同，但可以保证所观察到的数据存储区随着时间的推移会越来越新。具体而言，如果客户端发出一个读操作，稍后再发出针对同一对象的另一个读操作，那么，第二次读操作返回的将是相同的值或更晚写入的值。

读自身写 (read my writes) 是一种属性，也适用于由单个客户端执行的一系列操作。它保证由客户端完成的所有写操作的结果，对该客户端的后续读操作而言都是可见的。如果客户端向一个数据对象写入一个新值，然后再读取这个对象，那么，读操作将会返回该客户端最后写入的值（或者是由另一个不同的客户端在此之后写入的其他值）。对于没有发出过写操作的客户端来说，这种保证与最终一致性保证相同。（注：以前的文章中称之为“读取你的写入”，¹¹ 不过为了更准确地描述这种一致性保证，我对其从客户端的角度进行了重新命名。）

最后这四种读保证都是最终一致性的一种形式，但要比云存储系统通常提供的最终一致性模型更强。每种一致性保证的“强度”，不依赖于数据何时以及如何和服务端间传播，而是取决于一个读操作所允许的结果集大小。较小的读结果集表明较强的一致性。当要求强一致性时，只能返回一个结果，即已被写入的最新值。对于已经更新过多次的对象，最终一致性读操作可能会返回众多适合值中的一个。所有四个中间级别的一致性保证，彼此没有强弱之

分。也就是说，对于某个读操作，每种保证可能会有一组不同的响应。在某些情况下，应用程序可能需要请求多种保证，如稍后所示。例如，客户端可以同时请求单调读和读自身写两种保证，以便它看到的数据存储区与其自身进行的操作保持一致。¹¹

本文用于记录棒球得分的数据存储区是一个典型的键-值存储区，由“非 SQL(noSQL)”操作进行填充。写入，也称 *put*，修改与某个给定键相关联的值。读取，也叫 *get*，返回某个键所对应的值。但这些一致性保证也可应用于采用其他读写操作类型的其他类别的复制数据存储系统，如文件系统和关系型数据库。这就是为什么这些一致性保证由写入（而不是数据值）进行定义的原因。例如，某个系统提供了增量操作或追加操作，那么，对对象进行的所有写入都可能对该对象的观测值构成影响，而不仅仅是最新的写入。此外，尽管本文的示例并不要求原子更新，这些保证也适用于那些访问多个对象的原子事务。

表 2 显示了各种一致性保证的典型性能和可用性。其中按照从劣到优分别对三个属性进行评级。一致性评级依据的是先前定义的一致性保证的强度。性能是指完成一个读操作所花的时间，即读取等待时间。可用性是指在服务器发生故障的情况下，一个读操作成功返回适当一致性数据的可能性。

从一致性的角度来看，强一致性是可取的，但其所提供的性能和可用性却是最差的，因为它通常需要读取指定主站或大部分副本。另一方面，最终一致性允许客户端从任意副本读取数据，但其提供的数据一致性最弱。性能与一致性之间呈负相关，这并不奇怪，因为较弱的一致性措施通常允许读请求被发送到更大范围的服务器。在有多个的数据足够新的服务器可供选择的情况下，客户端更有机会选中一台邻近的服务器。访问本地服务器与远程服务器的等待时间可能相差 100 倍。同理，较大的服务器选择余地，意味着客户端更容易找到一个（或通过定额配备一个）可达的服务器，从而获得更高的可用性。

每种一致性保证措施都提供了一致性、性能和可用性的一个独特组合。表 2 中每个单元的标签不是精确的科学（我可以就这个话题写上一整篇文章）。有人可能会说，标为“可”的一些项目实际上应该为“良”，反之亦然。的确，这些特征在某种程度上要取决于实现、部署和操作方面的细节。对于某些客户端，最终一致性读经常可能返回强一致性的结果，而且可能并不比强一致性读效率更高。^{3, 13} 但各种一致性保证措施之间的一般性比较，从定性上来看还是准确的。表 2 给出的信息至少说明，在选择采用特定一致性模型的特定数据复制方案时，人们需要进行仔细权衡。

表 2. 一致性、性能和价值权衡。

保证	一致性	性能	可用性
强一致性	优	劣	劣
最终一致性	劣	优	优
一致性前缀	可	良	优
限定过期法	良	可	劣
单调读	可	良	良
读自身写	可	可	可

图 1. 一场简化的棒球赛。

```
Write ( "visitors" , 0);
Write ( "home" , 0);
for inning = 1 ..9
  outs = 0;
  while outs < 3
    visiting player bats;
    for each run scored
      score = Read ( "visitors" );
      Write ( "visitors" , score + 1);
  outs = 0;
  while outs < 3
    home player bats;
    for each run scored
      score = Read ( "home" );
      Write ( "home" , score + 1);
end game;
```

虽然未提供任何证据，但我断言所有这些一致性保证均可成为同一存储系统的选项。实际上，我和同事已经在 MSR 硅谷实验室创建了此类系统的一个原型（但这是另一篇文章要讨论的话题）。在我们的这个系统中，要求不同一致性保证措施的客户端在执行读操作时，会有不同的性能和可用性体验，即使在访问共享数据时也是如此。让我们假定存在一个这样的存储系统，可以让客户端选择这六种读保证措施。下面我继续演示如何在使用这些措施 ... 以棒球为例。

以棒球赛为应用样例

对于不熟悉棒球，但喜欢读代码的读者，图 1 说明九局棒球赛的基本知识。比赛开始比分为 0-0。客队首先击球，然后连续击球，直到他们一方三人出局为止。随后主队击球，直到三人出局。比赛共进行九局。当然，这里省去了对于像我一样狂热的棒球爱好者来说非常重要的很多细节。不过，

图 2. 一个比赛示例的写序列。

```
Write ( "home" , 1)
Write ( "visitors" , 1)
Write ( "home" , 2)
Write ( "home" , 3)
Write ( "visitors" , 2)
Write ( "home" , 4)
Write ( "home" , 5)
```

对于本文来说，所有必要的东西的确都已解释清楚了。

假设比赛得分记录在两个对象中的一个键-值存储区，一个是“客队”得分的跑垒数，一个是“主队”得分的跑垒数。当一个队跑垒得一分时，读操作读取其当前得分，将返回值加 1，再将新值写回键-值存储区。

作为一个具体例子，请参看比赛示例的写入日志，如图 2 所示。在这场比赛中，主队先得一分，然后客队将比分扳平，接着主队连得两分，这样依次进行下去。

该序列写入的值可能来自棒球比赛单行得分结果，各局比分如图

3 所示。这场假设的比赛现处于第七局中间（众所周知的第七局暂停），主队以 2-5 比分领先。

假设存有客队和主队总得分的键-值存储区驻留在云中，并在若干服务器间复制。不同的读保证措施可能导致客户端在比赛进行当中读到不同的得分。表 3 列出了读取客队和主队分数时，六种一致性保证措施中每一种措施可能返回的完整得分集合。注意：客队分数排在前面，所有可能出现的不同返回值用逗号隔开。

强一致性读只能返回一个结果，即当前比分，而最终一致性读可能返回 18 种可能比分中的一种。注意：由一对最终一致性读操作返回的很多比分结果，事实上从来没有在比赛中出现过。例如，读取客队得分时，可能会返回 2，而读取主队得分时，可能会返回 0，尽管主队在比赛中从未落后过。一致性前缀属性将比分结果限制为某段时间内真实存在过的比分。限定过期读操作所能返回的结果，明显取决于所期望的限定值。表 3 给出了限定为一局时的可能比分，也就是说，比分结果最多晚一局；当限定为七

图 3. 该比赛示例的单行得分。

	1	2	3	4	5	6	7	8	9	跑垒数
客队	0	0	1	0	1	0	0			2
主队	1	0	1	1	0	2				5

表 3. 对应每种一致性保证的可能读取的所有比分。

强一致性	2-5
最终一致性	0-0, 0-1, 0-2, 0-3, 0-4, 0-5, 1-0, 1-1, 1-2, 1-3, 1-4, 1-5, 2-0, 2-1, 2-2, 2-3, 2-4, 2-5
一致性前缀	0-0, 0-1, 1-1, 1-2, 1-3, 2-3, 2-4, 2-5
限定过期法	最多迟一局的比分：2-3, 2-4, 2-5
单调读	在读取 1-3 之后：1-3, 1-4, 1-5, 2-3, 2-4, 2-5
读自身写	对于数据写入者：2-5 对于数据写入者以外的任何人：0-0, 0-1, 0-2, 0-3, 0-4, 0-5, 1-0, 1-1, 1-2, 1-3, 1-4, 1-5, 2-0, 2-1, 2-2, 2-3, 2-4, 2-5

图 4. 记分员的角色。

```
score = Read ( "visitors" );
Write ( "visitors", score + 1);
```

局或更多局时，在本例中，其结果集与最终一致性的结果集相同。实际上，一个系统不太可能以“局”为单位来表示过期时间。因此，本例假设数据读取者请求 15 分钟的时间限定，而上一局正好持续了这么长时间。对于单调读，可能的返回值取决于过去读过哪些数据。对于读自身写，则取决于谁正在向键-值存储区写入数据；在本例中，假设所有的写操作均由单独一个客户端完成。

参与者的读需求

现在，让我们看看想知道一场棒球赛比分的各种人对数据一致性的需求。当然，每个人都可以通过执行强一致性读，来检索客队和主队的得分。正如我们在上一节所指出的那样，在这种情况下，只会有一种可能值被返回，即：当前比分。但如表 2 所示，要求强一致性的数据读取者，可能会经历更长的响应时间，甚至可能发现他们所请求的数据，因服务器临时故障或网络中断而变得不可用。本节的宗旨是评估每个参与者所需的最低一致性需求。这些客户端通过请求比强一致性弱的读保障措施，有可能体验到更好的性能和更高的数据可用性。

官方记分员。 官方记分员负责通过将比分写入持久性的键-值存储区，来维护比分。图 4 说明记分员在每次客队得分时采取的步骤；记分员在主队得分时采取的行动与此类似。注意：此代码为图 1 所示整个棒球赛代码的一个片段。

记分员的读操作需要什么一致性措施呢？毫无疑问，记分员需

图 5. 裁判的角色。

```
if first half of 9th inning complete then
  vScore = Read ( "visitors" );
  hScore = Read ( "home" );
  if vScore < hScore
    end game;
```

要读取最近的一次得分，然后将得分加 1，生成新的得分。否则，记分员就会冒险写入一个不正确得分，破坏整场比赛，还有可能激起愤怒球迷骚乱。假设主队此前已得五分，而且刚刚拿下第六分。如果采用最终一致性读，可能会返回从零到五的任何一个分值，如表 3 所示。也许，记分员非常幸运，读到了正确的得分，但这可不一定。

有趣的是，尽管记分员需要强一致性的数据，但他并不需要进行强一致性的读操作。由于记分员是唯一更新比赛得分的人，他可以请求读自身写保障措施，从而获得与强一致性读同样的效果。从本质上讲，记分员利用特定应用的知识，来获得较弱的一致性读所带来的好处，而不必真正地放弃一致性。

这种差别似乎非常小，但实际上在实践中可能非常重要。存储系统在处理强一致性读的过程中，必须悲观地假定，来自世界任何地方的客户端都可能刚刚更新过数据。因此，系统必须访问大部分服务器（或一组固定的服务器），以确保客户端提交的读操作访问的是最新写入的数据。另一方面，系统在提供读自身写保证的过程中，只需记录客户端先前执行的写操作集合，并找到已完成所有这些写操作的服务器。¹¹ 在棒球赛中，之前得到的分数，也就是之前由记分员完成的写操作，可能在几分钟，甚至是几小时之前就已经发生。在这种情况下，几乎任何服务器都已收到之前的写入操作，而且能够应答下一个要求读自身写保证的读请求。

图 6. 电台体育记者的角色。

```
do {
  vScore = Read ( "visitors" );
  hScore = Read ( "home" );
  report vScore and hScore;
  sleep (30 minutes);
}
```

裁判。 裁判是指在本垒板后主持棒球赛的人。裁判在大多数情况下并不真正关心比赛的当前比分。只有一个例外，就是在第 9 局上半场之后，也就是客队已击完球，主队正准备击球的那个时刻。由于这是最后一局（每个队都不能得负分），主队如果比分领先的话，就表明已经赢了；因此，在某些比赛中，主队可以跳过最后一局的击球。需要作此决定的裁判所对应的代码如图 5 所示。

在第 9 局进行中，裁判要读取得分时，的确需要读取当前得分。否则，如果他错以为主队领先，就可能会过早地结束比赛，或者让主队进行不必要的击球。与记分员不同，裁判从来不会写得分；他只是读取由官方记分员写入的值。因此，为了获得最新信息，裁判必须采用强一致性读。

电台记者。 在美国的大部分地区，电台会定期宣布正在进行的或已结束的比赛的比分。例如，旧金山地区的 KCBS 电台每隔 30 分钟就会播报一次体育新闻。电台记者完成图 6 所示的步骤。一个类似的，也许更为现代的例子是，当观众正在观看 ESPN 时，电视屏幕底部滚动显示体育比赛成绩。

如果电台记者播报的不是最新的比分，也没有问题。人们都已习惯收听旧闻。因此，对于他的

图 7. 体育新闻记者的角色。

```
While not end of game {
    drink beer;
    smoke cigar;
}
go out to dinner;
vScore = Read ( "visitors" );
hScore = Read ( "home" );
write article;
```

图 8. 统计员的角色。

```
Wait for end of game;
score = Read ( "home" );
stat = Read ( "season-runs" );
Write ( "season-runs", stat +
score);
```

图 9. 统计监察员的角色。

```
do {
    stat = Read ( "season-runs" );
    discuss stats with friends;
    sleep (1 day);
}
```

读操作，某种形式的最终一致性也是可以的。哪种一致性保证是电台记者所希望的呢？

如表 3 所示，采用最弱保证的读，即最终一致性的读，可能会返回从来不存在的比分。对于图 3 给出的单行得分示例，此种读操作可能会返回一个客队 1-0 领先的比分，尽管客队实际上从未领先过。电台记者不希望播报这种虚假的比分。因此，记者希望他两次都从快照中读取数据，快照中包含由记分员写入的值的一致性前缀。这使得记者能够读到之前某个时间的实际比分，但不必是当前比分。

然而，仅读一致性前缀还不够。对于图 3 所示的单行得分，

记者可能读到一个 2-5 的比分，即当前比分，等过了 30 分钟，又读到一个 1-3 的比分。例如，如果记者碰巧从主服务器上读取数据，而后又从可能位于远程数据中心的另一台服务器上读取数据，而该服务器已从主站上断开，尚未接收最新的写入数据，那么就可能出现上述情况。由于大家都知道棒球比分是单调递增的，如果在随后的新闻报道中依次播出 2-5 和 1-3 的比分，会让人们觉得记者非常愚笨。如果记者除了要求一致性前缀之外，还要求单调读保证，那么就可以避免出现这种情形。请注意：两种保证措施任中一种都不能独立满足要求。

另外，记者请求限定期限不超过 30 分钟的限定过期读，配合单调读，可以得到同样的效果。这将确保记者观察到的比分最多过时 30 分钟。由于记者每 30 分钟才读取一次数据，他接收的比分一定越来越新。当然，记者可以要求更紧的限定，比如五分钟，以便获得具有合理及时性的比分。

体育新闻记者。另一个有趣的人物就是体育新闻记者。他在观看比赛后，要写一篇报导发表在晨报上或上传到某个网站。不同体育新闻记者的表现可能也不相同，但通过观察（我也当过体育新闻记者），我发现他们的行为通常如图 7 所示。

体育新闻记者可能并不急着写他的文章。在这个例子中，他悠闲地外出吃晚饭，然后坐下来对比赛进行总结。他肯定要保证

他报导的比分是比赛的最终正确得分。因此，他想得到强一致性读的效果。不过，他并不需要支付这笔开销。如果体育新闻记者知道他在比赛结束后花了一个小时吃晚饭，那么他也一定知道，距离记分员最后更新比分至少也有一个小时了。因此，为限定过期读设置一小时的限定，足以保证体育新闻记者能够读到最终比分。在实际应用中，任何服务器都应该能回应此类读操作。实际上，最终一致性读很可能在一个小时后返回正确的比分，但是唯一能够让体育新闻记者 100% 确定自己得到最终比分的方法，就是请求采用限定过期法。

统计员。球队统计员负责跟踪球队和每个球员各赛季的统计数据。例如，统计员可能计算本赛季她的球队的得分总数。假设这些统计数据也保存在持久性的键-值存储区。如图 8 所示，主队统计员在每场比赛结束后的某个时间，将得分加到上个赛季得分总数上，然后将这个新值写入数据存储区。

当统计员读取当天的球队得分时，她需要确信自己得到的是最终比分。因此，她需要执行一个强一致性读操作。如果统计员在比赛结束后等待一段时间，那么，限定过期读也可能实现同样的效果（正如前面针对体育新闻记者的讨论）。

当读取本赛季当前的统计数据时，也就是图 8 所示的第二个读操作，统计员也希望采用强一致性读操作。如果返回的是旧统计数字，那么，写回的更新值会令球队的总得分减少。由于统计员是唯一将统计数据写入数据存储区的人，因此，她可以采用读自身写保证，以获得最新值（如前面讨论的那样）。

统计监察员。定期检查球队赛季统计数据的其他人员通常会满

表 4 棒球赛参与者的读保证方式。

官方记分员	读自身写
裁判	强一致性
电台记者	一致性前缀与单调读
体育新闻记者	限定过期法
统计员	强一致性、读自身写
统计监察员	最终一致性

足于最终一致性。统计数据每天仅更新一次，略显过时的数字还可以接受。例如，一个球迷询问本赛季他的球队的总得分，就可以通过最终一致性读，得到一个合理的答案，如图9所示。

结论

显然，存储棒球赛得分并非云存储系统的杀手级应用。对于从一个简单例子得出结论，我们应当谨慎。但也许我们可以从中得到一些教训。

对于上一节讨论过的各类棒球赛参与人员所希望采取的一致性保证，表4进行了总结。记住：所列举的一致性措施并非唯一可接受的措施。具体而言，每个参与者采用强一致性是没有问题的，但是，放松对一致性读的要求，很可能会获得更好的性能和更高的可用性。此外，存储系统可以更好地做到跨服务器读工作负荷均衡，这是因为，在选择用于应答弱一致性读请求的服务器方面有了更大的灵活性。

这些参与者可以看作是不同的应用程序，它们访问共享的数据：棒球赛比分。对于某些情形，如记分员和体育新闻记者，基于特定应用知识，数据读取者很清楚：即使他采用读自身写或限定过期保证方式发出弱一致性读请求，他也可以得到强一致性数据。对于另一些情形，如电台记者，多种保证方式必须组合起来使用，才能满足数据读取者的需求。对于其他情形，如统计员，希望采用不同的保证方式读取不同的数据对象。

我从本练习中得出了四个主要结论：

► **本文提出的所有六种一致性保证都非常有用。** 请注意：每种保证在表4中至少出现一次。除了一类客户端外，仅提供最终一致性的系统将无法满足其他所有类型客户端的需求；除了两类客户端

外，仅提供强一致性的系统可能会在其他所有情形下表现欠佳。

► **不同的客户端在访问相同的数据时，可能希望采用不同的一致性措施。** 系统经常会将一个具体的一致性绑定到一个特定的数据集或数据类。例如，人们普遍认为，银行数据必须是强一致性的，而购物车数据仅需要最终一致性即可。棒球赛的例子表明，人们所期望的一致性既决于数据类型，也取决于谁在读取数据。

► **即使是简单的数据库，各种用户也可能对一致性有不同的需求。** 棒球赛比分是可以想象的最简单的一种数据库，仅由两个数字组成。但这有效地阐释了不同一致性选项的价值所在。

► **客户端应能选择自己想要的一致性。** 系统可能无法预测或确定指定应用程序或客户端所需的一致性。首选的一致性选项往往取决于数据的使用方式。此外，在读最新的数据时，了解谁写入的数据，或者知道数据最后写入的时间，有时可能允许客户端执行松弛一致性读，从而获得相应的好处。

反对提供最终一致性方式的人们所持的主要论点经常是，它增加了应用程序开发人员的负担。虽然这可能是事实，但不必有过多的额外负担。第一步是定义开发人员可以理解的一致性保证；请注意，表1中的六种保证分别是用寥寥数语加以描述的。让存储系统依照严格顺序执行写操作，就可以使应用程序开发人员避免对并发写入所造成的更新冲突问题进行处理时所遇到的复杂性。这样开发人员的工作就只剩下选择自己想要的读一致性措施。要做出这种选择，就需要他们对其应用程序的语义有深入了解，但不必改变程序的基本结构。上一节提供的几个代码片段，都不需要增加额外的代码行来专门处理过时的数据。

仅提供强一致性的云存储系统，让开发人员能够很容易编写出

正确的程序，但可能会错失松弛一致性所带来的好处。一致性、性能和可用性之间固有的权衡是实际存在的，随着复制服务地理范围的扩大，问题有可能变得更加突出。这表明云存储系统至少应考虑提供更多的读一致性选择。有些云服务提供商已经提供强一致性和最终一致性两种读操作，但从文中可以看出，他们的最终一致性模型对应用程序来说可能并不理想。云存储客户端通过选择几种一致性保证，并从不同的副本读取数据，不仅可以让各种各样的应用程序受益，而且可以获得更高的资源利用率，并节约成本。

参考资料

1. Abadi, D. Consistency tradeoffs in modern distributed database system design. *IEEE Computer*, (Feb. 2012).
2. Amazon. Amazon DynamoDB; <http://aws.amazon.com/dynamodb/>.
3. Anderson, E., Li, X., Shah, M., Tucek, J. and Wylie, J. What consistency does your key-value store actually provide? In *Proceedings of the Usenix Workshop on Hot Topics in Systems Dependability*, (2010).
4. Bailis, P., Venkataraman, S., Franklin, M., Hellerstein, J. and Stoica, I. Probabilistically bounded staleness for practical partial quorums. In *Proceedings VLDB Endowment*, (Aug. 2012).
5. Brewer, E. CAP twelve years later: How the "rules" have changed. *IEEE Computer*, (Feb. 2012).
6. Calder, B. et. al. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings ACM Symposium on Operating Systems Principles*, (Oct. 2011).
7. Cooper, B., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.A., Puz, N., Weaver, D. and Yerneni, R. PNUTS: Yahoo!'s hosted data serving platform. In *Proceedings International Conference on Very Large Data Bases*, (Aug. 2008).
8. Google. Read Consistency & Deadlines: More Control of Your Datastore. Google App Engine Blog, Mar. 2010; <http://googleappengine.blogspot.com/2010/03/read-consistency-deadlines-more-control.html>.
9. Kraska, T., Hentschel, M., Alonso, G. and Kossmann, D. Consistency rationing in the cloud: Pay only when it matters. In *Proceedings International Conference on Very Large Data Bases*, (Aug. 2009).
10. Saito, Y. and Shapiro, M. Optimistic replication. *ACM Computing Surveys*, (Mar. 2005).
11. Terry, D., Demers, A., Petersen, K., Spreitzer, M., Theimer, M., and Welch, B. Session guarantees for weakly consistent replicated data. In *Proceedings IEEE International Conference on Parallel and Distributed Information Systems*, (1994).
12. Vogels, W. Eventually consistent. *Commun. ACM*, (Jan. 2009).
13. Wada, H., Fekete, A., Zhao, L., Lee, K. and Liu, A. Data consistency properties and the trade-offs in commercial cloud storages: The consumers' perspective. In *Proceedings CIDR*, (Jan. 2011).

Doug Terry (terry@microsoft.com) 是位于加州山景城 (Mountain View) 的微软研究院硅谷实验室首席研究员。

版权所有 / 作者所有。发表权已授予 ACM。\$15.00