Spanner, TrueTime 和 CAP 理论

Eric Brewer

VP, Infrastructure, Google

2017-02-14

Spanner 是 Google 的高可用的全球 SQL 数据库[CDE + 12]。它管理着大规模的复制的数据。大规模既指数据量方面，又有事务量方面。它为写入其中的每项数据分配全局一致的实时时间戳，客户端可以在整个数据库上执行全局一致的读操作，而无需使用锁（译注：因为 Spanner 使用的是物理时间，而且是全球分布的，所以这里的**全局**既可以理解为逻辑上的整体，也可以理解为全球性的）。

CAP 定理[Bre12]说，下面三个期望的属性中，你最多只能同时达到两个：

- C：一致性（Consistency），本文中我们可以认为这是指顺序一致性（Serializability）；
- A：读取和更新的 100％可用性（Availability）；
- P：对网络分区（Partitions）的容忍。

舍弃其中一个字母，就剩下三种系统：CA，CP 和 AP。请注意，并非自然就会有这三个属性中的两个，有许多系统只有其中的一个属性，甚至一个也没有。

对于“广域”上的分布式系统，通常认为网络分区是不可避免的，尽管不一定常见[BK14]。一旦你认为网络分区是不可避免的，任何分布式系统必须准备好放弃一致性（剩下 AP）或可用性（剩下 CP），这不是人们想做的选择。事实上，CAP 定理的出发点是让设计者认真对待这种权衡。但是有两个重要的警告：首先，你只需要在实际发生网络分区期间放弃去某些东西，即时那时也有有许多缓解措施（参见文章“CAP 理论 12 年回顾”[Bre12]）。其次，CAP 定理关注的是 100%可用性，而本文是关于现实的**高**可用性涉及的权衡（译注：**高**但不是 100%）。

Spanner 声称同时达到了 CA

尽管是一个全球分布式系统，Spanner 却声称具有一致性和高可用性，这意味着没有网络分区，因此很多人表示怀疑[1]。这是否意味着 Spanner 是 CAP 定义的 CA 系统？简短的答案是技术上“不是”，但效果上“是”，用户可以并确实认为是 CA 系统。

纯粹主义的答案是“否”，因为网络分区总是可能发生，事实上在 Google 也确实发生过。在网络分区时，Spanner 选择 C 而放弃了 A。因此从技术上来说，它是一个 CP 系统。我们下面探讨网络分区的影响。

考虑到始终提供一致性（C），Spanner 声称为 CA 的真正问题是，它的核心用户是否认可它的可用性（A）。如果实际可用性足够高，用户可以忽略运行中断，则 Spanner 是可以声称达到了“有效 CA”的。这并不意味着 100％的可用性（Spanner 目前和将来也不会提供），而是如 5 个或更多个“9”（即 $1/10^5$ 或更少的失效）。反过来，真正的试金石是，那些希望自己本身的服务高可用的用户，他们是否会编写处理运行中断异常的代码：如果他们没有编写这些代码，那么他们已经假设 Spanner 的高可用性了。基于大量的 Spanner 内部用户，我们知道他们认为 Spanner 是高可用的。

第二点是存在许多其它运行中断的原因，除了“生死与共”的 Spanner 之外，其它原因也会让用户的服务失效。我们实际上关心**差异化可用性**，即用户是否确实已经发现 Spanner 已停掉了。**差异化**可用性比 Spanner 的**实际**可用性还要高。也就是说，你必须真的听到大树倒下的声音才算是出了麻烦（译注：也就是说，Spanner 的短

[1] 虽然我在谷歌工作，但除了推动把 Spanner 和 TrueTime 提供给我们的云平台客户之外，我并没有参与这两个项目。我的想法是提供一个外部人士对 Spanner 的客观的观点。因为我喜欢这个系统，而且为 Google 工作，所以我承认会存在偏见。

暂不可用不一定会立即造成用户的系统不可用）。

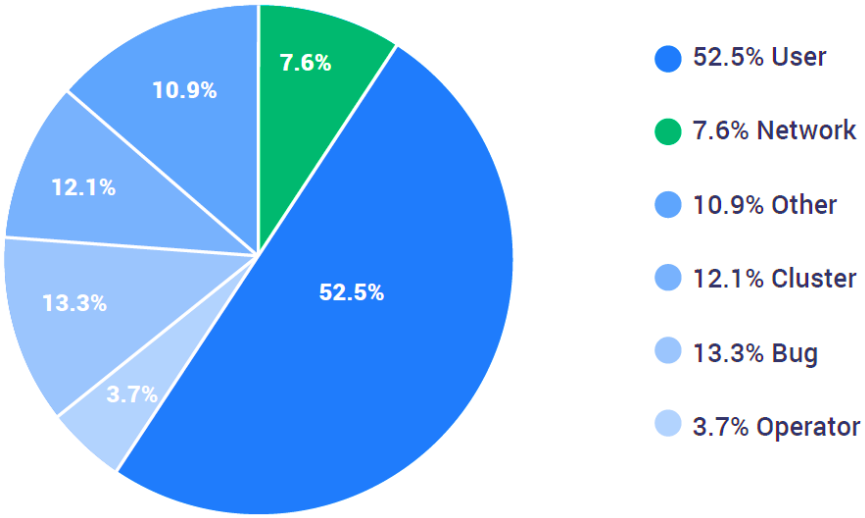
第三个问题是运行中断是否由于网络分区造成的。如果 Spanner 运行中断的主要原因不是网络分区，那么声称 CA 就更充分了。例如，任何数据库在所有副本都脱机的情况下都不能提供可用性，这与网络分区无关。这种多副本情况下的运行中断应该是非常罕见的，但如果网络分区的概率显著的更小，那么就可以有效地忽略网络分区对可用性的影响。对于 Spanner，这意味着可用性中断的发生，实际并非是由于网络分区，而是一些其它的多种故障（因为单一故障不会造成可用性中断）。

可用性的数据

在深入 Spanner 之前，值得先讨论一下 Chubby 的演进。Chubby 是另一个提供 CA 的广域系统。在 Chubby 的论文[Bur06]中提到 700 天中发生了 9 次 30 s 或更长时间的运行中断，其中 6 次与网络相关（如[BK14]中讨论的）。这对应的可用性最多也不超过 5 个 9，如果我们更贴近实际，假设每次中断平均有 10 min，那么就只有 4 个 9；如果每次中断有几个小时的话，就只有 3 个 9 了。

对于锁和一致性读/写操作，随着多项网络、架构和运维的改进，**目前的广域分布的 Chubby 集群能提供 99.99958%的平均可用性**（即仅有 30 s 多一点的运行中断）。从 2009 年开始，由于可用性“超额”，Chubby 的站点可靠性工程师（SRE）开始人为地强制定期中断服务（译注：即应急演习），以确保我们能发现对 Chubby 的依赖和其故障可能造成的影响。

在内部，Spanner 提供与 Chubby 水平相当的可靠性，少于 5.9 s。云版本与内部版本有相同的基础，但添加了一些新的部分，所以它的可靠性在短期内可能稍低一些。



上面的饼图显示了内部 Spanner 意外事件的原因。事件是意外的，但并非所有都严重到中断服务。一些事件可以轻易地处理掉。图中的数值是事件发生的频率而不是造成的后果。大量的事件（**用户事件**）是由于用户错误，例如超载或配置错误，并且大多只影响该用户，然而其它类别则可能影响区域中的所有用户。**集群事件**反映除网络外的底层基础架构问题，如服务器和电源的问题。Spanner 通过使用其它副本自动地处理这些事件，但有时需要 SRE 参与修复不完整的副本。**运维事件**是由 SRE 引起的事故，例如配置错误。 **Bug 事件**意味着软件 bug 触发的问题，这可能导致或大或小不同范围的运行中断。两个最大的中断都是由同时影响了某个数据库的所有副本的软件 bug 造成的。**其它事件**是各种大多只发生一次的问题。

网络事件（低于 8%）是网络分区和网络配置问题。还没有发生过较大集群的网络分区事件，也没有发生过一个分区的少数一方超过 Spanner 的 Quorum 的情况。我们确实看到个别数据中心或区域与其它网络断开。我们还有一些错误配置，短时调低了带宽，还有一些与硬件故障相关的暂时的延迟。曾经有一个事件，其中某个方向的网络中断，导致一个奇怪的分区，必须通过关闭一些节点才能解决。到目前为止，网络事件没有造成过大规模的运行中断。

总而言之，要声称“有效 CA”，系统必须处于这种相对概率状态：1）至少它在实践中必须具有非常高的可用性，以便用户可以忽略异常；2）由网络分区造成的运行中断应只占很小一部分。Spanner 同时满足两者。

这就是网络

许多人认为，Spanner通过使用TrueTime可以绕过CAP。TrueTime是一个提供全局同步时钟的服务。TrueTime是非比寻常的，但为实现 CA，TrueTime 的作用并不显著。后面的小节会介绍 TrueTime。某种程度说，Spanner 的特别之处是，Google 私有的广域网，加上多年的运维改进，显著降低了网络分区的发生，从而使高可用性成为可能。

首先，Google 运行自己的私有全球网络。Spanner 并非在公共的互联网上运行 —— 实际上，Spanner 的每个数据包只流过 Google 控制的路由器和链路（不包括到远程客户端的任何边缘链路）。此外，每个数据中心通常至少有三个独立的光纤将其连接到私有的全球网络。因此确保任何两个数据中心之间有多条网络通路[2]。类似的，数据中心内的设备和链路也是冗余的。因此，通常的灾难性事件，如光纤被切断，不会导致网络分区或运行中断。

因此，网络分区的真正风险不是网络被切断，而是某些大范围的配置或软件升级同时破坏了多个链路。这是一个真正的风险，并且 Google 持续地努力防止和缓解这一风险。一般的策略是限制任何更新的影响范围（“爆炸半径”），以便我们不可避免地推送一个错误的变更后，它只破坏一部分链路或副本。我们在修复问题之前不会尝试任何其它变更。

虽然网络分区大大减少了，但光速是有限的。广域上的一致性操作的往返时间（RTT）下限仍比较大，洲际约有几十毫秒或更长。（光速约 0.5 ft/ns，若洲际为 1 000 mile 的距离，约合 5 000 000 ft，则最少需要 10 ms）。Google 将一个“区域”的范围限制在 RTT 2 ms 之内，以在延迟和容灾之间达到平衡。Spanner 通过尽可能的事务批量化来缓解延迟，但这并不能降低单个事务的延迟。对于读操作，延迟通常较低，这是由于全局时间戳和使用本地副本的能力（如下节所述）。

具有较弱一致性的模型可能具有较低的更新延迟。然而，如果距离不够远，就会存在一个持久性较低的窗口。因为在数据被复制到另一个站点之前，如果本地站点遭受了灾害，所有的数据都可能被彻底破坏掉（译注：弱一致性模型会推迟数据的跨站点复制操作以降低延迟）。

网络分区时会发生什么

为了理解分区，我们需要更多地了解一下 Spanner 的工作原理。和大多数 ACID 数据库一样，Spanner 使用两阶段提交（2PC）和严格的两阶段锁，以确保隔离性和强一致性。2PC 被称为“反可用性”协议[Hel16]，因为事务期间所有成员必须正常工作。为缓解这一问题，Spanner 的每个成员实际是一个 Paxos 组（译注：多个节点组成逻辑上的单个节点），即便 Paxos 组中某个节点宕机了，每个 2PC“成员”也是高可用的。每个分组也是数据放置和复制的基本单元。

前面提到，一般来说当发生网络分区时，Spanner 会选择 C 而非 A。在实践中，这是考虑到：

- 使用 Paxos 组来达成关于某个更新的共识；如果 Paxos Leader 由于网络分区而不能维持 Quorum，则更新被暂停，并且系统不可用（由 CAP 的定义）。最终，如果大多数成员可用的话，新的 Leader 就可能选举出来；
- 对跨组事务使用 2PC 还意味着组内成员的网络分区可以阻止提交。

在实践中最可能的结果是，网络分区的一侧满足 Quorum，并将继续运行，也许需要重新选举 Leader。因此，服务继续可用，但是另一侧分区的成员数较少，不满足 Quorum，它们的用户无法访问该服务。这个例子说明了差异化可用性的重要性：那些无法访问服务的用户可能会有其它更严重的问题，例如失去连接，也可能已经宕机了。这意味着构建在 Spanner 之上的多区域服务，即使在网络分区时也能相对良好地运行。存在比较小的可能性，Spanner 的某一部分会完全不可用。

只要所有事务相关的组都有 Quorum 选举的 Leader，并位于分区的同一侧，Spanner 中的事务就会正常运行。这意味着一些事务正常提交，有些事务则会超时，但它们总是满足一致性的。Spanner 的一个特性是，任何正常返回的读操作都是一致的，即使事务稍后终止了（由于超时在内的任何原因）。

除了常规事务之外，Spanner 还支持快照读，即读取过去特定时刻的数据值。Spanner 维护多个时间版本的值，每个版本都有一个时间戳，因此可以为快照读操作返回正确的版本。特别地，每个副本都知道生成快照的时

[2] 真正的需求是链接目标的可用性，而不是多个链接本身。

间，并且任何副本能以本节点直接回复该时间点之前的读操作（除非它太旧了，以至于已经被垃圾收集）（译注：一般需要从多个节点读取数据并验证多数节点间是否一致）。类似地，很容易同时跨多个组异步读取。快照读完全不需要锁。事实上，**只读事务**被实现为在当前时刻（在任何最新的副本上）的**快照读**。

因此，快照读对网络分区而言更加健壮。特别的，快照读能在以下情况下正常工作：

1. 对于发起读操作的一侧网络分区，每个组至少存在一个副本
2. 对于这些副本，读时间戳是过去的。

如果 Leader 由于网络分区而暂停（这可能一直持续到网络分区结束），这时第 2 种情况可能就不成立了。因为这一侧的网络分区上可能无法选出新的 Leader（译注：见下节引用的解释）。在网络分区期间，时间戳在分区开始之前的读操作很可能在分区的两侧都能成功，因为任何可达的副本有要读取的数据就足够了。

关于 TrueTime

通常，同步时钟可以用于避免分布式系统中的通信。Barbara Liskov 提供了一个不错的概述和多个示例[Lis91][3]。对于我们的目的，TrueTime 是一个误差有界但非 0 的全局同步时钟：它返回的是一个时间区间，能保证执行调用的真实时刻落在这个区间内。因此，如果两个区间不重叠，我们能明确地将调用按真实时间排序。但如果区间存在重叠，我们就无法给出这两个调用的顺序了（译注：即并发的）。

Spanner 的一个微妙的之处是它用锁来实现顺序一致性（Serializability），但它用 TrueTime 来实现外部一致性（External Consistency，接近于线性一致性，Linearizability）。Spanner 的外部一致性不变量（Invariant）是：对任何两个事务T_1和T_2（即使在地球的两端），

$$\text{如果} T_2 \text{在} T_1 \text{提交之后才开始提交，则} T_2 \text{的时间戳大于} T_1 \text{的时间戳。}$$

引自 Liskov [Lis91，第 7 节]：

> 同步时钟可以用来降低违反外部一致性的概率。假设主节点（Primary）拥有租约（Lease，即一段时间的所有权），需要考虑整个副本组。副节点（Backup）发送到主节点的每条消息相当于是对主节点的一个租约。如果主节点持有一个来自**次多数**（Sub-majority[4]）副节点的未到期租约，则主节点能以单节点进行读操作。...
>
> 该系统中的不变量是：每当主节点执行读取时，它持有来自多数副节点的一个有效租约。如果时钟不同步，这个不变量将不再成立。

Spanner 使用 TrueTime 作为时钟，以确保不变量成立。具体地，在提交期间，Leader 可能必须等待，直到它确定提交时间在过去（基于误差界限）。实践中，这种“提交等待”的时间并不太长，而且与（内部）事务通信并行地进行。一般来说，外部一致性需要单调增加的时间戳，“等待不确定性结束”也是一种常见的模式。

Spanner 旨在通过对当选的 Leader 使用可顺延的租约，来延长 Leader 的在位时间（通常为 10s）。如 Liskov 所讨论的，每次 Quorum 达成共识时（译注：可能是关于其它事项），租约就会被顺延，因为参与者刚刚验证了 Leader 是有效的。当 Leader 失效时，有两个选项：1）等待租约过期，然后选举新的 Leader；或 2）重启旧的 Leader，这可能更快些。对于一些故障，我们可以发出一个“最后一息”的 UDP 数据包来释放租约，这是一个优化，以使租约尽快到期。由于计划外故障在 Google 的数据中心中很少见，所以长期的租约是合理的。租约还确保时间在 Leader 之间都是单调增长的，并且在没有 Leader 的情况下，使副节点能够在租约有效期内继续提供读取服务。

然而，TrueTime 的真正价值在于它在一致性快照方面的能力。回顾一下，多版本并发控制系统（Multi-Version Concurrency-Control systems，MVCC）[Ree78]有悠久的历史，它将旧版本分开保存，从而允许读取过时的版本，而不考虑当前的事务活动。这是一个非常有用和被低估的特性：具体到 Spanner 上，快照是一致的（在抓取快照时），因此如果你的系统中的某个不变量成立，它在快照中也会成立。即使你不知道是什么不变量！基本上，快照是在持续不断的多个事务之间抓取的，并且反映截至此时的所有内容，当然不会超过这些内容。

---

[3] Spanner 论文的两个作者，Wilson Hsieh 和 Sanjay Ghemawat，在 20 世纪 90 年代初都是 Barbara Liskov 的研究生。本文作者也是。

[4] Sub-majority 是 majority 减去 1，即将 Leader 也计算在内的 majority。

如果没有事务一致的快照，则很难从过去的时刻重新开始，因为这种快照的内容可能反映的是未完成的事务，这种事务可能违反一些不变量或完整性约束。正是缺乏一致性，导致有时难以从备份恢复系统。特别是有可能出现数据损坏，需要手动修复[5]。

例如，考虑使用 MapReduce 对数据库执行大规模的分析查询。Bigtable 存储着旧版本的数据，时间在数据分片上是“锯齿状”的，这使得结果不可预测，有时不一致（特别是对于较近的数据）。在 Spanner 上，同一个 MapReduce 可以选择精确的时间戳，并获得可重复和一致的结果。

TrueTime 还使得跨多个独立系统抓取快照成为可能，只要它们使用（单调增加的）TrueTime 时间戳提交，对抓取快照的时间达成一致，并存储多个时间版本的数据（通常在日志中）。这不仅限于 Spanner：你可以实现自己的事务系统，然后确保在两个系统（或甚至k个系统）上一致的快照。一般来说，在这些系统上需要一个 2PC（同时持有锁）以就抓取快照的时间达成一致，并确认成功，但系统不需要就其它事项达成一致，甚至这些系统可能会有很大的差异。

你还可以使用时间戳做为工作流传递的令牌。例如，如果对系统进行更新，则可以将更新的时间戳传递到工作流的下一个阶段，以便可以确定系统时间是否在该事件之后。在网络分区的情况下，这可能是不成立的，在这种情况下，如果想要一致性，下一个阶段应该等待（或如果想要可用性就继续下去）。没有时间令牌，很难知道你是否需要等待。使用时间戳不是解决这个问题的唯一方法，但这种方法是优雅且健壮的，能够保证最终一致性（Eventual Consistency）。当不同的阶段没有约定规则且管理员不同时，这是特别有用的——因为双方可以在没有通信的情况下对时间达成一致[6]。

快照是关于过去的，但你也可以对未来达成一致。Spanner 的一项特性是，为实现模式变更，可以就未来的某个时刻达成一致。这允许暂存对新模式的变更，以便能够同时提供新旧两个版本。一旦就绪，就可以选择一个时刻，在所有副本上以原子的方式切换到新的模式上（也可以选择暂存之前的时刻，但那时你可能还没有准备好）。至少理论上，你可以执行一些未来的操作，如计划删除或可预见的变更。

TrueTime 本身可能受到网络分区的影响。时间的来源是 GPS 接收器和原子钟的组合，两者都可以通过它们自身的微小漂移来保持精确的时间（译注：微小漂移是保持时间同步的调节手段，并不是说这两种时钟源不稳定）。由于每个数据中心都有冗余的“Time Master”，因此网络分区的两侧很可能继续获取准确的时间。然而，各个节点需要与 Time Master 的网络连接，否则它们自己的时钟将偏移。因此，在网络分区期间，它们与 Time Master 的偏差会逐渐地增长，取决于本地时钟漂移的速率限值。基于 TrueTime 的操作，例如 Paxos Leader 选举或事务提交，必须等待一段时间，但操作仍能够完成（假设 2PC 及 Quorum 通信正常）。

结论

Spanner 合理地声称是一个“有效 CA”系统。尽管运行在广域上，但它总是一致的，并达到了大于 5 个 9 的可用性。与 Chubby 一样，同时达到 CA 在实践中是可能的，前提是像 Google 那样能控制整个网络，但这在广域上是罕见的。此外，还需要大量冗余的网络链路、处理相关故障的架构规划、以及非常细致的运维，特别是对于升级。如果还是不幸发生了运行中断，Spanner 选择一致性而不是可用性。

Spanner 使用两阶段提交来实现顺序一致性，它使用 TrueTime 实现外部一致性、无锁的一致性读取、以及一致性快照。

[5] 作为比较，ARIES [MHL + 92]中，快照是故意打乱（Fuzzy）的，但是可以在每个页面上重放日志以使该页恢复到期望的被中断的事务（和逻辑时间）。这对恢复很有效，但不适合在快照上运行分析（因为它是打乱的）。

[6] 注意，时钟同步需要后台通信，包括 GPS 和定期校正。

致谢

特别感谢 Spanner 和 TrueTime 的专家：Andrew Fikes，Wilson Hsieh，和 Peter Hochschild。 另外还要感谢 Brian Cooper，Kurt Rosenfeld，Chris Taylor，Susan Shepard，Sunil Mushran，Steve Middlekauff，Cliff Frey，Cian Cullinan，Robert Kubis，Deepti Srivastava，Sean Quinlan，Mike Burrows，和 Sebastian Kanthak。

参考文献

[BK14] P. Bailis and K. Kingsbury. The Network is Reliable, *Communications of the ACM*. Vol. 57 No. 9,

Pages 48-55. September 2014. Also: https://aphyr.com/posts/288-the-network-is-reliable

[Bre12] E. Brewer. CAP Twelve Years Later: How the "Rules" Have Changed, *IEEE Computer*, Vol. 45, Issue 2, February 2012. pp. 23--29. CAP 理论十二年回顾："规则"变了（译注：这是本文作者之前的一篇文章）

[Bur06] M. Burrows. The Chubby lock service for loosely-coupled distributed systems. *Proceedings of OSDI '06: Fourth Symposium on Operating System Design and Implementation*, Seattle, WA, November 2006.

[CDE+12] J. Corbett, J. Dean, et. al. Spanner: Google's Globally-Distributed Database. *Proceedings of OSDI '12: Tenth Symposium on Operating System Design and Implementation*, Hollywood, CA, October, 2012. 厦门大学计算机系 林子雨 老师的译文

[Hel16] P. Helland. Standing on Giant Distributed Shoulders: Farsighted Physicists of Yore were Danged Smart! *ACM Queue*, Vol. 14, Issue 2, March-April 2016.

[Lis91] B. Liskov. Practical Uses of Synchronized Clocks in Distributed Systems. *ACM Principles of Distributed Computing (PODC)*. Montreal, Canada, August 1991.

[MHL+92] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh and P. Schwartz. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. *ACM Transactions on Database Systems*, Vol. 17, No. 1, March 1992, pp. 94-162.

[Ree78] D. Reed. Naming and Synchronization in a Decentralized Computer System, PhD Dissertation, MIT Laboratory for Computer Science, Technical Report MIT-LCS-TR-205. October 1978 [See Section 6.3 for list of versions with timestamps]