# Spanner，TrueTime 和 CAP 定理

Eric Brewer, VP, Infrastructure, Google. 2017-02-14

https://research.google/pubs/pub45855/
译者：Ying ZHANG. 2017-03; 2021-04; 2025-06
https://ying-zhang.cn/time/2017-spanner-truetime-cap-cn.pdf

Spanner 是 Google 的高可用的全球 SQL 数据库 [CDE+12]。它管理着大规模的多副本数据。大规模既指数据体量，又指交易数量。它为写入其中的每项数据分配全局一致的物理时间戳（real-time timestamps），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。

CAP 定理 [Bre12] 说，下面三个期望的属性中，最多只能同时实现两个：

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

舍弃其中一个字母，可以得到三种系统：CA，CP 和 AP。请注意，并非自然就会获得这三个属性中的某两个，有许多系统只具有其中的一个属性，甚至一个也没有。

对于 “广域” 的分布系统，通常认为网络分区是不可避免的，尽管不常见 [BK14]。一旦认为网络分区是不可避免的，任何分布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 定理的初衷是让设计者认真对待这种权衡。有两个重要的警告：首先，只需要在实际发生网络分区时才会放弃某些东西，即便那时也有许多缓解措施（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，而本文是关于实际的**高**可用性所涉及的权衡。

## 1 Spanner 声称一致且高可用（CA）

尽管是一个全球分布系统，Spanner 却声称一致且高可用，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它是 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。网络分区时，Spanner 选择 C 而放弃了 A。因此技术上说，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）。如果实际可用性高到用户可以忽略服务中断的程度，那么 Spanner 声称 “实际上是 CA” 就是合理的。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 $10^{-5}$ 或更少的失效）。反过来，真正的试金石，是那些希望自身服务高可用的用户是否会编写处理运行中断的代码：如果他们没有编写这些代码，那么他们已经假设 Spanner 高可用了。基于大量的 Spanner 内部用户，我们知道他们认为 Spanner 是高可用的。

第二点是还有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。我们实际上关心的是**差异化可用性**，即用户自身处于可用状态（并正在发起请求）从而感知到 Spanner 不可用的情况。**差异化**可用性比 Spanner 的**实际**可用性还要高。也就是说，必须真的听到大树倒下的声音，才算是出了麻烦。

第三个问题是运行中断是否是由网络分区造成的。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

[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。我的初衷是提供一个局外人对 Spanner 的客观看法。因为我喜欢这个系统，又在 Google 工作，所以我承认会存在偏见。

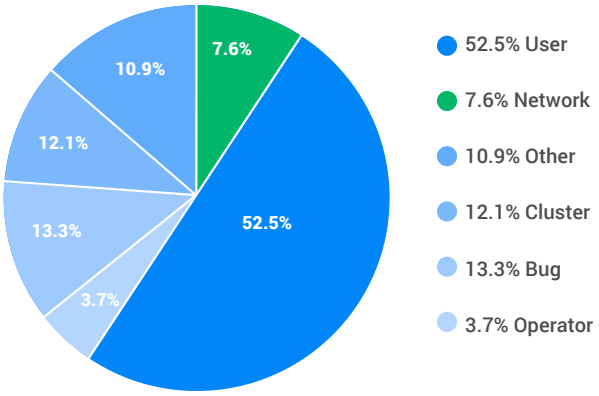
关。这种多副本情况下的运行中断应该是非常罕见的，但如果网络分区的概率显著的更小，那么就可以有效地忽略网络分区对可用性的影响。对于 Spanner，这意味着可用性中断的发生，实际并非是由于网络分区，而是由其他类型的多重故障所致。

## 2 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 的广域分布系统。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）。这对应的可用性不超过 5 个 9。如果我们更实际些，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。

对于锁和一致读/写操作，随着多项网络、架构和运维的改进，**广域分布的 Chubby 集群目前能提供 99.99958% 的平均可用性**（即仅约 30 s 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，以确保我们能发现 Chubby 故障可能造成的影响。

在内部，Spanner 提供与 Chubby 相当的可靠性，优于 5 个 9。云版本与内部版本有相同的基础，但添加了一些新的部分，所以短期内它的可靠性可能稍低一些。




上面的饼图显示了内部 Spanner 意外事故的原因。事故是意外的，但并非所有都严重到中断服务。一些事故可以轻易地处理掉。图中的数值是事故发生的频率而不是造成的后果。大量的事故（**用户事故**）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。**集群事故**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故，但有时需要 SRE 参与修复不完整的副本。**运维事故**是由 SRE 引起的事故，例如配置错误。**Bug 事故**意味着软件错误触发的问题，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数据库所有副本的软件 bug 造成的。**其它事故**是各种偶发问题。

**网络事故**类别（占比低于 8%）涵盖了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故，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。曾经有一个事故，其中单个方向的网络中断，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，网络事故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。

## 3 这就是网络

许多人认为，Spanner 通过使用 TrueTime 可以绕过 CAP。TrueTime 是提供全局同步时钟的服务。TrueTime 是不同寻常的，但为实现 CA，TrueTime 的作用并不显著。后面的小节会介绍 TrueTime。如果

说 Spanner 有什么特别之处，那其实是 Google 的广域网以及多年的运维改进，它们在实践中极大地减少了网络分区，从而实现了高可用性。

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路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，通常的灾难性事故，如光纤被挖断，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某条链路中断，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〔译注：光在光纤中的速率比在真空中慢〕）。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 通过广泛的事务流水线（pipelining）来缓解延迟，但这无助于降低单笔事务的延迟。对于读操作，延迟通常较低，这是由于全局时间戳和仅使用本地副本（见下节）。

具有较弱一致性的模型可能具有较低的更新延迟。虽然缓解了远距离往返延迟，但是引入了低持久性窗口。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：强一致要求同步复制，而弱一致性是异步复制。弱一致性协议在收到副本的确认消息之前，就答复客户端完成了数据复制。如果这时发生了严重事故，但是客户端误认为数据写入成功，不再重试，那么将导致数据丢失〕。

## 4 网络分区时会发生什么

为了理解分区，我们需要更深入了解 Spanner 的工作原理。和大多数 ACID 数据库一样，Spanner 事务使用两阶段提交（2PC）和严格的两阶段锁（2PL），以确保隔离和强一致。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 的每个事务成员实际是一个 Paxos 组，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；若 Paxos 主员由于网络分区不能维持 Quorum，则更新被暂停，且系统不可用（由 CAP 的定义）。如果大多数 Paxos 成员可用的话，最终新的主员仍能选出来；
- 对跨组事务使用 2PC 意味着事务成员间的网络分区可以阻止提交。

在实践中最可能的结果是，Paxos 组内的网络分区，一侧仍满足 Quorum，并将继续运行，也许需要重新选举主员。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，例如连接中断，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。Spanner 中多个分组不可用的可能性比较小。

只要事务相关的所有组都有 Quorum 选举的主员，并位于分区的同一侧，Spanner 中的事务就会正常执行。这意味着一些事务能正常提交，有些事务则会超时，但它们总是一致的。Spanner 的一个特性是，任何正常返回的读操作都是一致的，即使事务稍后中止了（由于超时在内的任何原因）。

[2]真正的需求是链接目标的可用性，而不是多链接本身。

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。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间，并且任何副本能基于本地数据直接回复该时间点之前的读操作（除非它太旧了或已经被作为垃圾回收了）。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，快照读更能容忍网络分区。特别的，快照读能在以下情况正常工作：

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，并且
2. 对于这些副本，读时间戳是过去的。

如果主员由于网络分区而暂停（这可能一直持续到网络分区结束），那么第 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主员。在网络分区期间，时间戳早于分区开始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中有待读取的数据就可以了。

## 5 关于 TrueTime

通常，同步时钟可以用于避免分布系统中的通信。Barbara Liskov 提供了不错的介绍和多个示例 [Lis91][3]。对于我们的目的，TrueTime 是一个偏差有界但非 0 的全局同步时钟：它返回的是一个时间区间，能保证执行调用的物理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物理时间排序。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。

Spanner 的一个精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，接近线性一致性，Linearizability）。Spanner 的外部一致性不变式（Invariant）是：对任何两个事务 $T_1$ 和 $T_2$（即使在地球两端），

> 若 $T_2$ 在 $T_1$ 提交之后才开始提交，则 $T_2$ 的时间戳大于 $T_1$ 的时间戳〔译注：两个时间戳均指提交时刻〕。

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可能。主员（Primary）持有租约（Lease），由整个副本组承认。从员（Backup）发送到主员的每条消息都顺延了对主员的租约。若主员持有一个来自次多数（Sub-majority[4]）从员的未到期租约，则主员能单方面（即基于自身数据）响应读操作。
>
> … …
>
> 该系统中的不变式是：每当主员响应读操作时，它总是持有来自次多数从员的一个有效租约。如果时钟不同步，这个不变式将不再成立。”

Spanner 使用 TrueTime，以确保不变式成立。具体地，在提交期间，主员必须等待，直到它确定提交时间已经过去（基于偏差区间）。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 旨在通过对当选的主员使用可顺延的租约，来延长主员的在任时间（通常为 10 s）。如 Liskov 所述，每次 Quorum 达成共识时，租约就会被顺延，因为参与者刚刚验证了主员是有效的。当主员失效时，有两个选项：（1）等待租约过期，然后选举新的主员；或（2）重启旧的主员，这可能更快些。对于一些故

[3]Spanner 论文的作者中，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〔Barbara Liskov 是 2008 年度图灵奖得主〕。

[4]Sub-majority 是 majority − 1，即将主员也计算在内的 majority。

障，主员可以发出一个 “临终”UDP 数据包主动释放租约，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，对不同任期的主员，时间戳都是单调增长的，并且在没有主员的情况下，从员组成的 Quorum 能够在租约有效期内继续响应读操作。

然而，TrueTime 的真正价值在于它在一致快照方面的作用。回顾一下，多版本并发控制系统（Multi-version Concurrency-Control，MVCC）[Ree78] 有很长的历史，它分别保留旧版本，从而允许读取过时的版本，而不受当前的事务活动影响。这是一个非常有用却被低估的特性：具体到 Spanner 上，快照是一致的（在获取快照时），因此如果系统中的某个不变式成立，它在快照中也会成立。即使你不知道是什么不变式！基本上，快照是在接连不断的多个事务之间获取的，并且反映截至此时的所有内容。若没有事务一致的快照，则很难从过去的时刻重新开始，因为可能有未完成的事务，可能违反一些不变式或完整性约束。正是缺乏一致性，导致有时难以从备份数据中恢复。特别是，表现为需要手动修复的数据损坏[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（特别是对于较新的数据）。在 Spanner，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获取快照成为可能，只要它们使用（单调增加的）TrueTime 时间戳提交，对获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，在这些系统上需要执行 2PC 事务（同时持有锁）以就获取快照的时间达成一致，并**确认成功**，但系统不需要对其它事项达成一致，甚至这些系统可能会有很大的差异。

还可以使用时间戳作为工作流传递的令牌。例如，若对系统进行了更新，则可以将此更新事件的时间戳传递到工作流的下一个阶段，这样下一阶段就能够判断自己的状态是否已经反映了该更新事件。在网络分区的情况下，这可能不成立，在这种情况下，如果想要一致性，下一个阶段就应该等待（如果想要可用性，就继续执行）。没有时间令牌，很难知道是否需要等待。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（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，但也可以对未来达成一致。Spanner 的一项特性是，为实现表模式变更，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表模式上（也可以选择暂存之前的时刻，但那时可能还没有准备好）。至少理论上，可以执行一些未来的操作，如计划删除或更改可见性。

TrueTime 本身可能受到网络分区的影响。时间的来源是 GPS 接收机和原子钟的组合，两者都可以通过它们自身保持精确的时间（但也有微小的漂移）。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。基于 TrueTime 的操作，例如 Paxos 主员选举或事务提交，必须等待更长时间，但操作仍能够完成（假设 2PC 及 Quorum 通信正常）。

## 6 结论

Spanner 有理由声称是一个 “有效 CA” 系统。尽管广域运行，但它总是一致的，并达到了优于 5 个 9 的可用性。与 Chubby 一样，同时实现 CA 在实践中是可能的，前提是像 Google 那样能控制整个网络，但这在广域范围很少见。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，尤其是升级。即使这样也会发生运行中断，这种情况下，Spanner 会选择一致性而不是可用性。

[5]作为比较，ARIES [MHL+92] 中，快照是有意乱序（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（因为它是乱序的）。

[6]注意，时钟同步需要后台通信，包括 GPS 和定期校正。

Spanner 使用两阶段提交（2PC）来实现顺序一致；它使用 TrueTime 实现外部一致、无锁的一致读以及一致快照。

## 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.

[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]